情報数理シリーズ

B-2

チューリングマシンと計算量の理論

守屋悦朗著

培風館

「情報数理シリーズ」企画委員会

『情報数理シリーズ』発刊にあたって

近年の情報化社会の進展は目覚しいものがあり，21 世紀がいわゆる「情報」の時代であることは言をまたない．特に，我が国は各種の資源をもたず，知識集約的な産業をその主体とせざるをえない．このためには，時代の変化に柔軟に対応でき，創造力に富む情報分野の人材の育成が急務である．

書店には，各種のソフトウェアを利用するための解説書が数多く並んでいる．特に，Windows 95 の出現以降，この傾向は増す一方である．しかし，これらのほとんどは欧米諸国で開発されたものであり，真の知識集約的産業国家を形成するには，欧米諸国で開発されたソフトウェアを使いこなすだけでは不十分である．基本ソフトウェアから高度な応用ソフトウェアまでを創り出していく必要がある．

一方，情報に関する教育は理工系の各学科で広く開講されている．しかし，その講義用テキストは，情報処理技術やプログラミング言語の修得を目的としたものが多く，必ずしも情報教育に必須の基礎とその応用を含めた本来あるべき講義内容を指向したものとはなっていないのが実情といえる．本シリーズは，電気・電子・通信・情報・経営の各系など広い意味で情報関連の学部生・大学院生，情報関連企業の技術者・研究者を対象としている．情報教育の講義内容を情報数理の面からとらえ，また，学部と大学院のカリキュラムの連携を考慮している．これらを反映し，本シリーズは次のような視点から構成した．

1. 基礎・基礎と応用・入門の三つのグループに大別する．
2. セメスタ制にあわせた簡単な記述とし，現行および将来のカリキュラムと対応づけたものとする．
3. 基礎理論と先端的な応用技術とのバランスをとる．

なお，本シリーズは主として早稲田大学大学院理工学研究科で情報関係の講義を担当している教員を執筆陣としている．専攻によらず，情報分野を横断的にとらえており，必ずしも体系的・網羅的な構成を目指したものではないことをお断りしておきたい．

最後に，本シリーズをまとめるにあたっては，企画の当初から（株）培風館の児玉 晴男，木村 博信両氏のご尽力をいただいた．ここに，心より感謝する．

平成 8 年 3 月　　　　企画委員会

序

計算量の理論はまだ若い．人間が計算 (アルゴリズム) というものについて理論的にきちんと考え始めたのは今からわずか数十年前の 1930 年代のことである．最初は純粋に数学的興味から研究が始まった計算 (アルゴリズム) についての研究が，1940 年代になってコンピュータが発明されるに至り，俄然現実的な面からも注目を浴びることとなった．そして，1950–60 年代には，例えば形式言語・オートマトン理論とかプログラム理論など，コンピュータにまつわるさまざまな新しい理論が誕生した．計算量理論は，この時代の豊かな実りをもとに，1970 年代に入ってやっときちんとした理論として姿を現したものである (1930 年以前には，アルゴリズムとは何かのきちんとした数学的定義さえなかった．アルゴリズムの数学的な定式化ができた後も 1960 年代までは，アルゴリズムが存在するかどうか (決定可能性) だけが関心の的であり，その複雑さにはさほどの関心が払われなかった)．以後，コンピュータの性能の進歩と歩調を合わせるかのように，多くの有能な研究者達によって研究は進展した．ところが実は，もっとも基本的な最重要問題の多くが現在でも依然として未解決のままである．例えば，$\mathbf{DL} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$ という計算量の階層 (問題を計算の難しさの程度で分類したもの) はどの包含関係 $\subseteq$ も真 ($\subsetneq$) であると予想されていながら，未だにそれが証明できないでいる．このように，計算量理論はまだまだ多くの有能な (とくに，若い) 研究者の力を必要としている．本書がそんな研究者を生むきっかけになってくれれば著者としてはこの上ない喜びである．

本書は，著者が早稲田大学，東京女子大学で使った講義ノートに加筆したものである．本書を書くにあたって，巻末や脚注に記したような多くの著書や論文を参考にさせていただいた．邦書も，すぐれた研究者によって書かれたものがすでにたくさん出版されており，本書が単に屋上屋を重ねるだけにならないことにもっとも気を配ったが，それが達成されていないとすればそれは著者の力量不足のせいである．具体的には，計算量理論全般を広く浅く概観することを目指した．また，歴史的背景や理論の現状などについてもできるだけ述べ，定義や定理の羅列に終わらないように心掛けたつもりである．本書を出発点として，計算量理論という未だ未知にあふれる山にさらに分け入って欲しい．

普通，計算のモデルとしては“チューリングマシン (チューリング機械)”を使うが，それは初学者に計算量理論をとっつきにくく感じさせる要因の1つでもある．巻末に掲げたように，コンピュータ世代の若者にとって馴染みの深い“プログラム”を基に分かりやすい計算量理論を展開しようとしたすぐれた著作もある．しかし，著者があえてチューリングマシンにこだわったのは，形式的な計算のモデルとしてはチューリングマシンがやはり最もすぐれていると考えたからである．本文中でも述べているように，歴史的に多くの計算モデルが提案されたが，現在でもチューリングマシンが最も広く用いられていることがそのことを証明しているように思うし，本書を出発点としてさらに先を目指す読者がそれらの論文や著書を読むときのことを考慮したからでもある．

本文中の各所に問を置いたが，これらは学んだことをその場で確かめるための，ごくやさしい問題ばかりであるから必ず解いて欲しい．章末の演習問題は，演習としての意味合い以外に，ページ数の制約があって本文中で取り上げることができなかった項目を補足するという役割も担っている (難問は一切なく，ややむずかしい問題にはヒントを付けた)．証明を省いた定理は「命題」としたが，必ず参考文献を示した．

本書は，著者自身がすべて TEX (LATEX) により (TEX のありがた迷惑なやり方を宥めすかしつつ) 入力編集した．したがって，読みやすさに関して問題があるとすれば著者の責任である．ただし，図のうち1・2・7章は LATEX の picture 環境で著者自身が描いたが，他は培風館にお願いして Mac で描いたものである．

最後に，講義を受講しノートをとったり誤りを指摘してくれたりした学生諸君に感謝したい．培風館の児玉晴男・木村博信両氏には予定より原稿が遅れご迷惑をおかけした．お詫びするとともに，ページ数やレイアウト等に関して著者のわがままを許していただいたことを感謝したい．

一旦脱稿の後，読み直して補足・修正を行なったら大幅にページ数が増えてしまった．その後さらに読み直してみたところ，まだ多数の誤りが見つかったし，まだまだ不十分だとも感じたが，完璧を期そうとすると修正作業が永久に収束しそうにない．その意味で，本書は未完成のままである (計算量の理論自体，現在大きな発展途上にある)．読者諸氏のご叱正を乞いたい．

1997 年 7 月

守 屋 悦 朗

目　次

1
準　備

この章では，次章以降随所で必要になる離散数学やアルゴリズムに関する基本的事項 (関係の代数，グラフ，論理，プログラムによるアルゴリズムの記述法，O 記法など) について概説する．この章は最初は読み飛ばし，後で必要になるたびに参照するとよい．

1.1　アルファベット・語・言語

記号 (symbol) や文字 (letter) を元とする有限集合を**アルファベット** (alphabet) という．アルファベットを表わすのに大文字のギリシャ文字を使うことが多い．Σ をアルファベットとするとき，Σ の元を重複を許して有限個並べた列

$$a_1a_2\cdots a_n \quad (a_1, a_2, \ldots, a_n \in \Sigma)$$

を Σ 上の**記号列**とか，**文字列** (string) とか，**語** (word) という．$w = a_1a_2\cdots a_n$ を構成している記号の個数 n $(n \geq 0)$ をこの語の**長さ** (length) といい，$|w|$ で表わす．とくに $n = 0$ の場合には**空語** (empty word) といい，ギリシャ文字 ε で表わす．Σ 上の語の全体を Σ^* で表わし，$\Sigma^* - \{\varepsilon\}$ を Σ^+ で表わす．Σ^* の部分集合を Σ 上の**言語** (language) という．

例 1.1 (1) $\Sigma = \{0, 1\}$ とすると，

$$\Sigma^* = \left\{ \begin{array}{l} \varepsilon, \\ 0, 1, \\ 00, 01, 10, 11, \\ 000, 001, \ldots, 111, \\ \quad \cdots \end{array} \right\} \quad \begin{array}{l} \text{長さ 0 の語} \\ \text{長さ 1 の語} \\ \text{長さ 2 の語} \\ \text{長さ 3 の語} \\ \ \end{array}$$

であり，Σ^* は可算無限集合である．

(2) 次の集合 EVEN と PRIME は $\Sigma = \{0,1\}$ 上の言語であり，WcW は $\{a,b,c\}$ 上の言語である：

$$
\begin{aligned}
\text{EVEN} &= \{w \in \Sigma^* \mid |w| \text{ は偶数 }\} \\
\text{PRIME} &= \{w \in \Sigma^* \mid w \text{は素数を 2 進数で表わしたもの }\} \\
\text{WcW} &= \{wcw \in \{a,b,c\}^* \mid w \in \{a,b\}^*\} \quad \square
\end{aligned}
$$

上例で，wcw は語 w, c, w をこの順に並べてつなげた語を表わす．このような，語をつなげる演算を**連接** (concatenation) といい，明示する場合は $\cdot$ で表わす (例えば wcwは $w \cdot c \cdot w$ と表わす) が，誤解が生じない場合には省略してもよい．$\varepsilon w = w = w\varepsilon$ である．とくに，w を n 個連接した語を w^n で表わす．$w^0 = \varepsilon$, $w^1 = w$ である．$w = a_1a_2\cdots a_n$ (各 $a_i \in \Sigma$) に対して，文字の並び順を逆にした語 $a_n\cdots a_2a_1$ を w の**反転** (reversal) とか，**鏡像** (mirror image) といい，w^R で表わす．任意の語 x, y に対して $(xy)^R = y^Rx^R$ が成り立つ．

語 $w \in \Sigma^*$ が $w = xyz$ $(x, y, z \in \Sigma^*)$ と表わされるとき，x, y, z それぞれを w の**部分語** (subword) という．とくに，x を**接頭語** (**プレフィックス**，prefix), z を**接尾語** (**サフィックス**，suffix) という．

問 1.1 Σ は ℓ 個の文字を含むアルファベット，w は Σ 上の語で $|w| = m$ とする．次のものの個数を求めよ．

(a) Σ 上の長さ n の語 (b) w の接頭語 (c) w の部分語

Σ 上の言語 L, L' に対して，L と L' の**連接** (**積**ともいう) $L \cdot L'$ (または LL' と書く), L の n **乗** L^n, L の**反転** (**鏡像**) L^R, L の **Kleene 閉包** (Kleene closure) L^*, L の**正閉包** L^+ を以下のように定義する：

$$
\begin{aligned}
L \cdot L' &= \{x \cdot y \mid x \in L,\ y \in L'\} \\
L^n &= LL\cdots L \quad (n \text{ 個の } L \text{ の連接}) \\
L^R &= \{x^R \mid x \in L\} \\
L^* &= \textstyle\bigcup_{n \geq 0} L^n \\
L^+ &= \textstyle\bigcup_{n \geq 1} L^n
\end{aligned}
$$

ただし，$L^0 = \{\varepsilon\}$ と定義する．$L^1 = L$ である．w が語のとき，$\{w\}$ を単に w と略記することがある．したがって，$\{w\}^*$ も w^* と略記する．

例 1.2 (1) $\Sigma = \{a,b\}$, $A = \{a, ab, baa\}$, $B = \{\varepsilon, ab\}$ とすると，

$A \subseteq \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$, $B \subseteq \Sigma^0 \cup \Sigma^2$, $AB = \{a, ab, ba^2, a^2b, (ab)^2, ba^3b\}$

である．

(2) $1^* = \{1\}^* = \{1^n \mid n \geq 0\} = \{\varepsilon, 1, 11, 111, \ldots\}$．$1^n$ は数 1 ではなく，1 を n 個並べた語であることに注意する． $\square$

問 1.2 次のものを $\{0,1\}$ 上の言語として表わせ．
(a) $\{w \in \{0,1\}^* \mid w$は 1 をちょうど 3 個含む $\}$
(b) 自然数 (0 を含む) を 2 進数表現したものの全体，および，そのうちの偶数の全体，3 の倍数の全体

1.2　帰納法と再帰的定義

可算無限個の離散的対象を扱う際，**数学的帰納法** (mathematical induction, 単に**帰納法**ともいう) はきわめて強力な証明法である．実際，本書で述べる定理の多くは帰納法を使って証明される．可算無限個の対象に自然数を対応付け，証明すべきことを自然数 n に関する命題 $P(n)$ として表わす．$P(n)$ がすべての (あるいは，k 以上のすべての) 自然数 n について成り立つことを帰納法で示すには，**基礎ステップ** (basis) と**帰納ステップ** (inductive step) に段階分けする．基礎ステップで $P(0)$ (あるいは $P(k)$) が成り立つことを示し，帰納ステップでは $P(n)$ が成り立つものと仮定する (これを**帰納法の仮定** (inductive hypothesis) と呼ぶ) と $P(n+1)$ も成り立つことを任意の $n \geq 0$ (あるいは $n \geq k$ なる任意の n) について示す．アンダーライン部分を「n 以下の任意の m に対して $P(m)$ が成り立つ」としてもよい．本書における帰納法の証明では，帰納ステップにおいて帰納法の仮定を最初に明示することは必ずしもせず，$P(n+1)$ の証明の過程で $P(n)$ (あるいは $P(m)$) を「帰納法の仮定により」という述べ方で使う．

本書では 0 も自然数に含め，自然数すべてからなる集合を $\mathbf{N}$ で表わす．また，整数すべての集合を $\mathbf{Z}$ で表わす:

$$\mathbf{N} = \{0, 1, 2, \ldots\}, \quad \mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}.$$

集合 X の**濃度** (cardinality) を $|X|$ で表わす．X が有限集合の場合，$|X|$ は X に属す元の個数である．X の部分集合 Y の**特性関数** (characteristic function) とは，次のように定義される関数 $\chi_Y : X \to \{0,1\}$ である:

$$\chi_Y(x) = \begin{cases} 1 & x \in X\text{のとき} \\ 0 & x \notin X\text{のとき}. \end{cases}$$

例 1.3 集合 X の部分集合の全体を 2^X で表わし，X の**べき集合** (power set) という．X が有限集合ならば $|2^X| = 2^{|X|}$ であることを，$|X|$ に関する帰納法で示そう．

(基礎ステップ) $|X| = 0$ のとき，$X = \emptyset$ であり，その部分集合は $\emptyset$ 自身だけであるから $2^X = \{\emptyset\}$ である．よって，$|2^X| = 1 = 2^{|X|}$ が成り立つ．

(帰納ステップ) $|X| \geq 1$ のとき，a を X の任意の元とし $Y = X - \{a\}$ とおく．X の部分集合には a を元として含まないものと含むものがあり，それらは「Y の部分集合」と「Y の部分集合に a を加えたもの」である．すなわち，

$$2^X = 2^Y \cup \{A \cup \{a\} \mid A \in 2^Y\}$$

である．右辺の 2 つの集合は共通部分を持たず，$A, B \in 2^Y$，$A \neq B$ ならば $A \cup \{a\} \neq B \cup \{a\}$ であることに注意する．$|Y| < |X|$ であるから帰納法の仮定より $|2^Y| = 2^{|Y|}$ である．よって，$|2^X| = 2^{|Y|} + 2^{|Y|} = 2^{|Y|+1} = 2^{|X|}$ である．

以後，「基礎ステップ」，「帰納ステップ」という語句は明示しない． □

集合や関数や関係やプログラムも，帰納法と同様に定義することができる (プログラムの再帰的定義については 1.6 節参照)．例えば，アルファベット Σ に対し，集合 Σ^* と，Σ 上の語の長さ $|\cdot|$ を次のように同時に定義しよう．

(1) $\varepsilon \in \Sigma^*$, $|\varepsilon| = 0$.

(2) $x \in \Sigma^*$ かつ $a \in \Sigma$ ならば $ax \in \Sigma^*$, $|ax| = |x| + 1$.

(3) (1) および (2) で定まるものだけが Σ^* の元である．

このような定義の方法を**帰納的定義** (inductive definition) とか，**再帰的定義** (recursive definition) という．条件 (3) (**限定句**と呼ぶ) がないと Σ^* は一意的に定まらないことに注意する．しかし，再帰的定義においては (3) のような条件は必ず必要なものであるから，いちいち明記することをせず省略することが多い．

問 1.3 $\Sigma = \{\langle, \rangle\}$ とする．Σ 上の言語 D を次のように再帰的に定義する (限定句は省略してある)．D のことを **Dyck 言語**という．

(1) $\varepsilon \in D$.

(2) $\alpha \in D \implies \langle\alpha\rangle \in D$.

(3) $\alpha, \beta \in D \implies \alpha\beta \in D$.

$P \Longrightarrow Q$ は「P ならば Q」を表わす．関数 $f : \Sigma^* \to \mathbf{N}$ を

$$f(x) = (x \text{ に現れる } \langle \text{ の個数}) - (x \text{ に現れる } \rangle \text{ の個数})$$

と定義する．$x \in D$ である必要十分条件は，$f(x) = 0$ かつ x の任意の接頭語 y に対して $f(y) \geq 0$ が成り立つことである．このことを証明せよ．

1.3 関係と順序

集合 X, Y の直積 $X \times Y$ の部分集合 R を **2 項関係** (binary relation) または単に**関係** (relation) という．また，R を X から Y への関係ともいい，$R : X \to Y$

と書く．$R: X \to X$ のとき，R は X の上の関係という．$(x, y) \in R$ のとき，x と y は「Rの関係にある」といい，xRy とも書く．

関数は関係の特別の場合である．まず，X の任意の元 x に対して $(x, y) \in R$ となる Y の元 y がたかだか1つだけ存在する場合，$R: X \to Y$ を X から Y への**部分関数** (partial function) という．とくに，任意の x に対して $(x, y) \in R$ となる y が必ず存在する (したがって，ちょうど1つ存在する) 場合，R を**全域関数** (total function) とか，単に**関数** (function) とか，**写像** (mapping) という．R が (部分) 関数の場合，$(x, y) \in R$ を $R(x) = y$ と書く．

$R: X \to Y$, $S: Y \to Z$ のとき，

$$S \circ R = \{(x, z) \mid (x, y) \in R \text{ かつ } (y, z) \in S \text{ を満たす } y \in Y \text{ が存在する}\}$$

で定義される関係 $S \circ R: X \to Z$ を R と S の**合成** (composition) という．

R を X 上の関係とする．各自然数 n に対して，関係 R^n を次のように定義する:

$$R^0 = \{(x, x) \mid x \in X\}, \quad R^{n+1} = R^n \circ R \ \ (n \geq 0).$$

このとき，$R^1 = R$ である．また，

$$R^* = \bigcup\nolimits_{n \geq 0} R^n, \quad R^+ = \bigcup\nolimits_{n \geq 1} R^n$$

によって定義される関係 R^*, R^+ をそれぞれ R の**反射推移閉包** (reflexive transitive closure)，**推移閉包** (transitive closure) という．定義より，

$$\begin{aligned} xR^*y &\iff xR^ny \text{ となる } n \geq 0 \text{ が存在する} \\ &\iff x = y \text{ または } xR^+y, \\ xR^+y &\iff xR^ny \text{ となる } n \geq 1 \text{ が存在する} \end{aligned}$$

が成り立つ．$P \iff Q$ は，P と Q が同値であること，すなわち P が成り立つのは Q が成り立つとき，かつそのときに限ることを表わす．

一方，xR^ny については，次のことが成り立つ．

定理 1.1 R を X 上の関係，x, y を X の元とする．xR^ny が成り立つ必要十分条件は

$$(*) \quad x = z_0,\ z_0Rz_1,\ z_1Rz_2,\ \ldots,\ z_{n-1}Rz_n,\ z_n = y$$

を満たす $z_1, z_2, \ldots, z_{n-1} \in X$ が存在することである．$(*)$ を

$$x = z_0 \, R \, z_1 \, R \, z_2 \ \cdots \ z_{n-1} \, R \, z_n = y$$

と略記する．

証明　n に関する帰納法で証明する．

$n = 0$ のとき $(*)$ は $x = y$ を表わしている．一方，定義より $xR^0y \iff x = y$ であるから，$n = 0$ のとき定理は成り立つ．

次に，$n \geq 0$ のとき定理が成り立つと仮定する．R^{n+1} の定義から，

$$
\begin{aligned}
xR^{n+1}y &\iff x(R^n \circ R)y \\
&\iff xRz_n \text{ かつ } z_nRy \text{ となる } z_n \in X \text{ が存在する}
\end{aligned}
$$

である．xR^nz_n であるから，帰納法の仮定により $x = z_0\,R\,z_1\,R\,z_2\,\ldots\,z_{n-1}\,R\,z_n$ を満たす $z_1, z_2, \ldots, z_{n-1} \in X$ が存在する．よって，$n+1$ のときも定理は成り立つ． □

例 1.4 (1) あるシステムにおいて，そのシステムがとりうる状態すべての集合を X とする．システムの状態が a から b へ 1 単位時間で変化することを $a \rightsquigarrow b$ と書くことにすると，$\rightsquigarrow$ は X 上の関係である．上述の結果より，$a \rightsquigarrow^* b$, $a \rightsquigarrow^+ b$, $a \rightsquigarrow^n b$ はそれぞれ a から b へ「0 単位時間以上で」，「1 単位時間以上で」，「ちょうど n 単位時間で」変化することを表わす．

(2) $\mathbf{N}$ の上の関係 R を $nRm \overset{\text{def}}{\iff} n = m+1$ で定義すると，任意の k に対して $R^k = \{(m+k, m) \mid m \in \mathbf{N}\}$ であり，

$$
\begin{aligned}
nR^km &\iff n = m + k \quad (k \in \mathbf{N}) \\
nR^*m &\iff nR^km \text{ となる } k \geq 0 \text{ が存在する} \iff n \geq m \\
nR^+m &\iff nR^km \text{ となる } k \geq 1 \text{ が存在する} \iff n > m
\end{aligned}
$$

である． □

R を X 上の関係，x, y, z を X の任意の元とする．次の 3 つの性質を満たすとき，R を**半順序** (patial order) あるいは単に**順序**という．

(i) xRx **反射律** (reflexive law)

(ii) xRy かつ $yRz \implies xRz$ **推移律** (transitive law)

(iii) xRy かつ $yRx \implies x = y$ **反対称律** (antisymmetric law)

半順序を表わす記号として $\leq$ や $\sqsubseteq$ などがよく用いられる．$\leq$ が X 上の半順序であり，かつ X の任意の 2 元 x, y に対して $x \leq y$ または $y \leq x$ が必ず成り立つならば，$\leq$ は**全順序** (total order) あるいは**線形順序** (linear order) であるという．半順序関係が定義されている集合を**半順序集合** (partially ordered set) といい，全順序関係が定義されている集合を**全順序集合** (totally ordered set) という．順序を明示するときは，$(X, \leq)$ のように表わす．$x \leq y$ かつ $x \neq y$ であるとき $x < y$ と書く．

例 1.5 (1) 整数の間の大小関係 $\leq$ は $\mathbf{Z}$ 上の全順序である．

(2) 集合 X の部分集合の間の包含関係 $\subseteq$ は 2^X 上の半順序である ($|X| \geq 2$ ならば全順序ではない)． □

問 1.4 半順序であることを示せ.

(a) 自然数の間の関係 $|$: $n \mid m \overset{\text{def}}{\Longleftrightarrow} n$ は m を割り切る

(b) アルファベット Σ 上の語の間の関係 $\preceq$: $x \preceq y \overset{\text{def}}{\Longleftrightarrow} x$ は y の接頭語

$(X, \leq)$ を半順序集合, A を X の部分集合とする. A の任意の元 a に対して $a \leq x$ であるような $x \in X$ を, A の**上界** (upper bound) という. 上界の最小値 (存在すれば) を**上限** (least upper bound) という. 同様に, **下界** (lower bound), **下限** (greatest lower bound) が定義される. 例えば, 実数の集合 $\mathbf{R}$ において, 半開区間 $[1, \infty)$ は開区間 $(0,1)$ の上界すべてからなる集合であり, $(0,1)$ の上限は 1, 下限は 0 である. 関数 $f : X \to Y$ (Y は半順序集合) の上界, 上限等は, その値域 $f(X)$ の上界, 上限等のことである.

アルファベット Σ の元の間に全順序 $\leq$ が定義されているとする.

$$
\begin{aligned}
x \sqsubseteq y \quad \overset{\text{def}}{\Longleftrightarrow} \quad & x \text{ は } y \text{ の接頭語であるか，または} \\
& x = uav,\ y = ubw,\ a < b \text{ となる } a, b, u, v, w \text{ が存在する} \\
& (a, b \in \Sigma;\ u, v, w \in \Sigma^*)
\end{aligned}
$$

で定義された Σ^* 上の関係 $\sqsubseteq$ を**辞書式順序** (lexicographic order) という. 例えば, $\Sigma = \{a, b\}$, $a < b$ のとき, $\varepsilon, a, b, aa, ab, aab, aabb, abab$ を辞書式順序で昇順に並べると

$$\varepsilon,\ a,\ aa,\ aab,\ aabb,\ ab,\ abab,\ b$$

である.

問 1.5 $\leq$ を X 上の全順序とする. X の任意の部分集合が $\leq$ に関して最小元を持つならば, $\leq$ を**整列順序** (well order) という. 辞書式順序は整列順序でないことを示せ.

アルファベット Σ 上の語に対して次のように定義される順序 $\lll$ を**標準順序** (standard order) という:

$$
\begin{aligned}
x \lll y \quad \overset{\text{def}}{\Longleftrightarrow} \quad & |x| < |y| \text{ であるか，または} \\
& |x| = |y| \text{ かつ 辞書式順序のもとで } x \sqsubseteq y.
\end{aligned}
$$

標準順序は整列順序である.

さて, R を X 上の関係とする. X 上の関係で, (関係を $X \times X$ の部分集合とみたとき) R を含み, 推移律 (反射律と推移律) を満たすようなもので包含関係に関して最小のもの, を R の**推移閉包** (**反射推移閉包**) という. すなわち, 次の条件を満たすとき, S を R の推移閉包 (反射推移閉包) という.

(i) $R \subseteq S$.

(ii) Sは推移律 (反射律と推移律) を満たす.

(iii) $R \subseteq S'$ かつ S' が推移律 (反射律と推移律) を満たす $\Longrightarrow$ $S \subseteq S'$.

すでに R^+, R^* をそれぞれ R の推移閉包，反射推移閉包と呼んだ理由は次の定理にある．

定理 1.2 R^+, R^* はそれぞれ R の推移閉包，反射推移閉包である．

証明 まず，R^+ が R の推移閉包であることを示そう．上記の (i)~(iii) が成り立つことを示す．

(i) $R \subseteq R^+$ であることは明らか．(ii) 次に，$(x,y) \in R^+$, $(y,z) \in R^+$ とすると，R^+ の定義より $(x,y) \in R^p$, $(y,z) \in R^q$ となる正整数 p, q が存在する．よって，$(x,z) \in R^q \circ R^p = R^{p+q}$ であり，これは $(x,z) \in R^+$ を意味する．ゆえに，R^+ は推移律を満たす．

(iii) 最後に，$R \subseteq S'$ かつ S' が推移律を満たすとする．$R^+ \subseteq S'$ であることを示すために，任意の自然数 $n \geq 1$ に対して $R^n \subseteq S'$ であることを帰納法で示す．$n = 1$ のときは，$R^1 = R$ であるから仮定 $R \subseteq S'$ より明らか．$(x,y) \in R^{n+1} = R^n \circ R$ とすると，$(x,z) \in R$, $(z,y) \in R^n$ となる z が存在する．帰納法の仮定から $R^n \subseteq S'$ であり，したがって，$(x,z) \in S'$ かつ $(z,y) \in S'$ である．ところが S' は推移律を満たすから，このことより $(x,y) \in S'$ であることが導かれる．これで $R^{n+1} \subseteq S'$ であることが証明された．

R^* についても同様である． □

1.4 グラフ

集合 V と $V \times V$ の部分集合 E との対 $G = (V, E)$ を**有向グラフ** (directd graph) という．G は V 上の 2 項関係に他ならない．V の元を**頂点** (vertex), E の元を**辺** (edge) という．V, E をそれぞれ $V(G)$, $E(G)$ と表わすことがある．$v \in V$ に対して

$$\begin{aligned} \text{in-deg}(v) &= |\{u \in V \mid (u,v) \in E\}| \\ \text{out-deg}(v) &= |\{w \in V \mid (v,w) \in E\}| \end{aligned}$$

と定義し，それぞれ v の**入次数** (in-degree)，**出次数** (out-degree) という．

有向グラフ $G = (V, E)$ は次のように図示される．まず，頂点すべてを平面上に点として描く．次に，$(u,v) \in E$ であるならば u から v へ向かって向きのある線を描く．例えば，図 1.1 は

$$G = (\{a,b,c,d,e\},\ \{(a,b),\ (a,c),\ (a,d),\ (b,d),\ (c,d),\ (d,b),\ (d,d)\})$$

を図示したものである．(d,d) のような辺を**自己ループ** (self-loop) と呼ぶ．

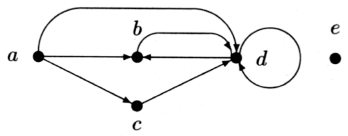


図 **1.1**　有向グラフ

有向グラフでは辺に向きがある((u, v) と (v, u) が区別される)のに対し，辺に向きのないグラフを無向グラフといい，次のように定義される．集合 V と，ちょうど2個の元からなる V の部分集合の集合 E との対 $G = (V, E)$ を**無向グラフ** (undirected graph) という．V の元を**頂点**といい，E の元を**辺**というのは有向グラフと同じである．辺 $\{u, v\}$ と $\{v, u\}$ とは同じものであり，誤解のおそれのない限り，これらも有向グラフの辺と同様に (u, v) または (v, u) で表わす(uv または vu で表わすこともある)．また，2頂点間に2本以上の辺を許すこともあり，そのようなグラフを**多重グラフ** (multigraph) とか，**多辺グラフ**という．

$$\deg(v) = |\{u \mid (u, v) \in E\}|$$

を v の**次数** (degree) という．図1.2の左図は無向グラフ $G = (\{a, b, c, d, e, f\}, \{(a, b), (a, c), (a, d), (b, c), (b, e), (c, d), (c, e), (c, f), (d, e), (e, f)\})$ を図示したものである．右図は多重グラフである．

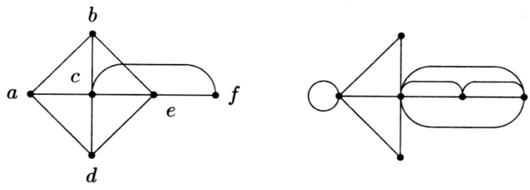


図 **1.2**　無向グラフと多重グラフ

$V(G_1) \subseteq V(G)$, $E(G_1) \subseteq E(G)$ であるとき，グラフ G_1 はグラフ G の**部分グラフ** (subgraph) であるという．また，$V(G)$ から $V(G_2)$ への全単射 φ が存在し $(u, v) \in E(G) \Longleftrightarrow (\varphi(u), \varphi(v)) \in E(G_2)$ であるとき，G と G_2 は**同型** (isomorphic) であるという．すべての頂点間に辺が存在する無向グラフ G を**完全グラフ** (complete graph) といい，$|V(G)| = n$ のとき K_n で表わす．

有向(あるいは無向)グラフ G における頂点 u から頂点 v への**道** (path) とは，$V(G)$ の元の有限列

$$p = \langle v_0, v_1, \ldots, v_n \rangle$$

で，各 i $(0 \le i < n)$ に対して $(v_i, v_{i+1}) \in E(G)$ であるものをいう．n を p の

長さ (length) という．$\langle v \rangle$ は v から v への長さ 0 の道である．すなわち，道とは辺に沿ってグラフの頂点をたどった経路のことである．道 p の始点 v_0 と終点 v_n が一致している長さ 3 以上の道を**閉路** (circuit) という．とくに，すべての v_i $(0 \leq i < n)$ が異なる閉路を**サイクル** (cycle) という．また，すべての頂点をちょうど 1 回だけ通る閉路を**ハミルトン閉路** (Hamilton circuit) といい，ハミルトン閉路を持つグラフを**ハミルトングラフ**という．

問 1.6 すべての辺をちょうど 1 回通る閉路が存在するグラフを**オイラーグラフ** (Euler gragh) という．G がオイラーグラフであるための必要十分条件は，G が連結で，どの頂点の次数も偶数であることである．このことを証明せよ．ただし，**連結** (connected) とは，どの頂点間にも道が存在することである．

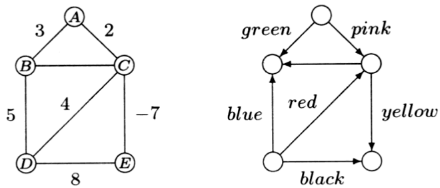


図 1.3 ラベル付きグラフ

グラフの頂点や辺に，ある特定の集合の元 (**ラベル**と呼ぶ) を付けたものを**ラベル付きグラフ** (labeled graph) という．形式的には，ラベル付きグラフは，グラフ $G = (V, E)$ とラベル付け関数 $\lambda_V : V \to L_V$，$\lambda_E : E \to L_E$ とで定まる．L_V，L_E はそれぞれ頂点と辺に付けるラベルの集合であり，$\lambda_V(v)$，$\lambda_E(e)$ はそれぞれ頂点 v と辺 e に付けられたラベルを表わす．頂点だけ，あるいは辺だけにラベルを付けることもある．辺に付けられたラベルが数値の場合には，**重み付きグラフ** (weighted graph) ということもある．ラベル付きグラフの例を図 1.3 に示した．

閉路を持たない有向グラフ $T = (V, E)$ で次の性質を持つものを**根付き木** (rooted tree) あるいは単に**木** (tree) という:

(i) 入次数が 0 の頂点がちょうど一つある．この頂点を**根** (root) という．

(ii) 根以外の頂点はすべて入次数が 1 である．

(iii) 任意の頂点 x に対し，根から x への道がちょうど 1 つだけ存在する．

実は，上記の条件は (i), (ii) だけで十分で，条件 (iii) は条件 (i), (ii) から導かれる．木の頂点は**ノード** (node) とも呼ばれる．出次数が 0 のノードを**葉** (leaf) という．$(u, v) \in E$ のとき，u を v の**親** (parent)，v を u の**子** (child) とい

う．共通の親を持つノードは**兄弟** (sibring) であるという．u から v への道があるとき，u を v の**先祖** (ancestor)，vを u の**子孫** (descendant) という．すべてのノードは根の子孫である．ノード u とその子孫すべてからなる木を，u を根とする T の**部分木** (subtree) という．根からノード u への道の長さを u の**深さ** (depth) とか**レベル** (level) という．葉の深さの最大値を木の**高さ** (height) といい，$\mathrm{height}(\cdot)$ で表わす．木を図で表わすときは，図 1.4 に示したように，根を上方に描き，辺に矢印を付けないのが普通である．この例では，A が根，G, H, I, J, K が葉，高さは 3 である．

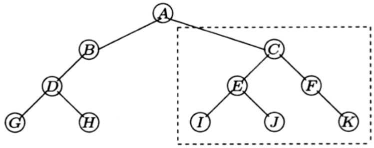


図 1.4　2 分木

兄弟の間に順序が定まっている木を**順序木** (ordered tree) という．ここでは順序木をきちんと定義することはしないが，図で描いたときに左から右に順序がついているものとみなす．どのノードの出次数も 2 以下である順序木を **2 分木** (binary tree) という．2 分木においてノード v に 2 つの子があるとき，図的に左にあるものを**左の子** (left child)，右にあるものを**右の子** (right child) と呼び，左の子 (右の子) を根とする部分木を**左部分木 (右部分木)** という．図 1.4 の木は A を根とする 2 分木である．A の左の子は B，右の子は C である．B には右の子がなく，F には左の子がない．破線で囲んだ部分は，A の右部分木である．

問 1.7 木の定義において，条件 (iii) は (i), (ii) より導かれることを示せ．

1.5　論理式とブール関数

値 0 または 1 を**真理値** (truth value) あるいは**論理値**といい，真理値を値にとる変数を**論理変数** (logical variable) とか，**ブール変数** (boolean variable) という．真理値として，0,1 の代わりに **false** (偽), **true** (真) を用いることもある．真理値上で定義された次の 6 つの演算 ($\neg$ は単項演算，それ以外は 2 項演算) を総称して**論理演算** (boolean operation) という．

x	y	$\neg x$	$x \vee y$	$x \wedge y$	$x \to y$	$x \leftrightarrow y$	$x \oplus y$
0	0	1	0	0	1	1	0
0	1	1	1	0	1	0	1
1	0	0	1	0	0	0	1
1	1	0	1	1	1	1	0

真理値や論理変数を論理演算記号で結んでできる式を**論理式** (logical formula) または**ブール式** (boolean formula) という (ここでは厳密な定義を省略するが，普通の数式と同様に考えればよい)．例えば，x, y, z を論理変数とするとき，0, x, $x \vee y$, $(\neg x) \vee (x \wedge y)$, $\neg(x \vee y \vee z) \to (0 \to 1)$ などは，それぞれ論理式である．論理演算の結合の強さは $\neg$, $\wedge$, $\vee$, $\oplus$, $\to$, $\leftrightarrow$ の順であると定め，誤解の恐れのない限り論理式の中の括弧を省略してもよい．例えば，$(\neg x) \vee (x \wedge y)$ は $\neg x \vee x \wedge y$ でよいが，$\neg(x \vee y \vee z) \to (0 \to 1)$ の括弧は省略できない．$\vee, \wedge, \oplus, \leftrightarrow$ はいずれも結合律を満たすので，例えば $x \vee y \vee z$ において括弧は不要であるが，$\to$ は結合律を満たさないので $x \to (y \to z)$ の括弧は必要である．

問 1.8 $\vee, \wedge, \oplus, \leftrightarrow$ のいずれも (任意の一つを$\circ$とする) 結合律を満たす (すなわち，任意の x, y, z に対して $(x \circ y) \circ z = x \circ (y \circ z)$ が成り立つ) ことを示せ．また，$\to$ は結合律を満たさないことを示せ．

$\neg x$ を x の**否定** (negation) といい，$x \vee y$ を x と y の**論理和** (or あるいは disjunction)，$x \wedge y$ を x と y の**論理積** (and あるいは conjunction) という．$x \vee y$ は x または y の少なくとも一方が 1 であることを意味するのに対し，$x \oplus y$ は x と y のどちらか一方だけが 1 であることを意味する．そのため，$\oplus$ のことを**排他的論理和** (exclusive or) という．$x \to y$ は「$x = 1$ ならば $y = 1$ である」ことを，$x \leftrightarrow y$ は「x と y の真理値は等しい」ことをそれぞれ表わす．言い換えれば，$x \to y$ は「x が成り立つならば y が成り立つ $(x \Longrightarrow y)$」ことを意味し，$x \leftrightarrow y$ は「x と y は同値である $(x \Longleftrightarrow y)$」ことを意味する．

論理式 f が論理変数 $x_1, \ldots, x_n$ を含んでいるとき，f は $x_1, \ldots, x_n$ の関数

$$f : \{0,1\}^n \to \{0,1\}$$

とみることができ，$f(x_1, \ldots, x_n)$ とも書く．このような場合，f を n 変数の**ブール関数** (boolean function)，各 x_i を**ブール変数**という．

任意の $x_1, \ldots, x_n \in \{0,1\}$ に対して $f(x_1, \ldots, x_n) = 1$ であるとき，f を**トートロジー** (tautology) という．定義より，$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$ が任意の $x_1, \ldots, x_n$ に対して成り立つことと，$f(x_1, \ldots, x_n) \leftrightarrow g(x_1, \ldots, x_n)$ がトートロジーであることとは同値である．$f(x_1, \ldots, x_n) = 1$ となる $x_1, \ldots, x_n \in \{0,1\}$ が存在する場合，$f(x_1, \ldots, x_n)$ は**充足可能** (satisfiable) であるという．充足可能でないとき，**充足不能** (unsatisfiable) であるという．

定理 1.3 任意の x, y, z に対して，次の等式が成り立つ．

(1) $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$, $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

(2) $\neg(x \vee y) = \neg x \wedge \neg y$, $\neg(x \wedge y) = \neg x \vee \neg y$

(3) $x \to y = \neg x \vee y$

(4) $x \leftrightarrow y = (x \to y) \wedge (y \to x)$

証明 (4) だけを示す (他も同様である)．x, y の値の組合わせすべてについて，部分論理式の値を論理演算記号の下に記した．このような表を**真理表** (truth table) という．左右両辺の値 (太字のところ) はいずれの場合も等しい． □

x	y	$x \leftrightarrow y$	$(x \to y)$	$\wedge$	$(y \to x)$
0	0	**1**	1	**1**	1
0	1	**0**	1	**0**	0
1	0	**0**	0	**0**	1
1	1	**1**	1	**1**	1

問 1.9 定理 1.3 の (1)〜(3) および次の恒等式を証明せよ．
$\neg(x \leftrightarrow y) = x \oplus y = (x \vee y) \wedge (\neg x \vee \neg y) = (x \wedge \neg y) \vee (\neg x \wedge y)$

論理変数，または論理変数の否定 ($\neg$ を付けたもの) を**リテラル** (literal) という．論理式 (ブール関数) f がリテラルの論理積の論理和 g として表わされるとき，g を f の**加法標準形** (**DNF**: disjunctive normal form) という．f がリテラルの論理和の論理積 h として表わせるとき，h を f の**乗法標準形** (**CNF**: conjunctive normal form) という．また，定数論理式 1 はトートロジーの加法標準形でありかつ乗法標準形でもあるとし，0 は充足不能論理式の加法標準形かつ乗法標準形であるとする．

定理 1.4 どんな論理式 $f(x_1, \ldots, x_n)$ にもその加法標準形と乗法標準形が存在する．

証明 加法標準形について証明する．以下の証明において $\wedge$ と $\vee$，0 と 1 を入れ替えれば乗法標準形についての証明が得られる．

$f(x_1, \ldots, x_n)$ の真理表を考える．これは 2^n 個の行からなっている．$1 \leq i \leq 2^n$，$1 \leq j \leq n$ に対してリテラル z_{ij} を，i 行目の x_j の値が 1 のとき $z_{ij} = x_j$，0 のとき $z_{ij} = \neg x_j$ と定める．i 行目の $(x_1, \ldots, x_n)$ の値に対して，$f(x_1, \ldots, x_n) = 1$ のとき $g_i = z_{i1} \wedge \cdots \wedge z_{in}$ と定義し，$f(x_1, \ldots, x_n) = 0$ のとき g_i は空であると定義する．最後に，$g = g_1 \vee \ldots \vee g_{2^n}$ と定義する．ただし，真理表のすべての行で $f(x_1, \ldots, x_n) = 0$ である場合，すなわち f が充足不能である場合には $g = 0$ とする．

$f = g$ であること，すなわち任意の $x_1, \ldots, x_n$ に対して $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$ であることを示そう．f が充足可能であるとする．真理表の i 行目の $(x_1, \ldots, x_n)$ の値 (それを $(x_{i1}, \ldots, x_{in})$ とする) に対して $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$ であることを示せばよい．

はじめに，$x_{ij} = 1$ ならば $z_{ij} = x_{ij} = 1$ であり，$x_{ij} = 0$ ならば $z_{ij} = \neg x_{ij} = 1$ であるから，任意の j に対して $z_{ij} = 1$ であることに注意する．一方，$k \neq i$ ならば，k 行目の $(x_1, \ldots, x_n)$ の中には i 行目の $(x_1, \ldots, x_n)$ と値が異なる箇所 (それを $x_{j'}$ とする) がある．したがって $z_{kj'} \neq x_{ij'}$ であるから，$z_{kj'} = 0$ である．すなわち，

$$z_{k1} \wedge \cdots \wedge z_{kn} = \begin{cases} 1 & k = i \text{ のとき} \\ 0 & k \neq i \text{ のとき} \end{cases}$$

である．

さて，$f(x_1, \ldots, x_n) = 1$ の場合，g_i の定義より g_i は空ではなく $g_i(x_1, \ldots, x_n) = 1$ である．よって，$g(x_1, \ldots, x_n) = \cdots \vee g_i(x_1, \ldots, x_n) \vee \cdots = 1$ である．次に，$f(x_1, \ldots, x_n) = 0$ の場合，g_i は空であり，$k \neq i$ ならば $g_k(x_1, \ldots, x_n) = z_{k1} \wedge \cdots \wedge z_{kn} = 0$ であるから $g(x_1, \ldots, x_n) = 0$ である．　□

例 1.6 定理 1.3 の証明の中に示されている真理表に従って $x \leftrightarrow y$ の加法標準形を求めると，

$$x \leftrightarrow y = (\neg x \wedge \neg y) \vee (x \wedge y)$$

である．また，乗法標準形は $(x \vee \neg y) \wedge (\neg x \vee y)$ である．　□

これまで述べてきた論理式は，正確には**命題論理** (propositional logic) と呼ばれる論理体系の論理式である．本書ではこれを**ブール式** (Boolean formula) と呼ぶ．ブール式を拡張して，**量限定ブール式** (quantified Boolean formula, **QBF** と略す) を次のように再帰的に定義する．同時に，QBF の中に現れる論理変数が**自由** (free) であるか**束縛**されている (bound) かも定義する．同じ変数であっても，現れる位置によって自由か束縛されているかが異なる．$f(x_1, \ldots, x_n)$ は f の中の自由変数が $x_1, \ldots, x_n$ であることを表わす．

(1) ブール式は QBF である．この中に現れる論理変数はすべて自由である．

(2) $f(x_1, \ldots, x_n)$ が QBF ならば，任意の i $(1 \leq i \leq n)$ に対して

$$(\forall x_i f(x_1, \ldots, x_n)) \text{ および } (\exists x_i f(x_1, \ldots, x_n))$$

はそれぞれ QBF である．これらの QBF に現れる x_i はすべて束縛されている．それ以外の変数は，f の中で自由であるか束縛されているかをそのまま受け継ぐ．

(3) (1), (2) で定まるものだけが QBF である．

$\forall$, $\exists$ は $\neg$ と同じ結合順位であると定め，$\forall$, $\exists$ の適用範囲を示す括弧は可能な限り省略してもよい．$\forall$ を**全称記号** (universal quantifier)，$\exists$ を**存在記号** (existential quantifier) という。直観的に言うと，$\forall x f(x)$ は「任意の x に対して $f(x)$ が成り立つ ($f(x)=1$ である)」ことを表わし，$\exists x f(x)$ は「$f(x)$ が成り立つ ($f(x)=1$ となる) ような x が存在する」ことを表わす．

$f(x_1,\ldots,x_n)$ を QBF とする (ブール式の場合を含む)．$f(a_1,\ldots,a_n)=1$ となる $a_1,\ldots,a_n \in \{0,1\}$ が存在するならば f は**充足可能**であるという．ただし，$f(a_1,\ldots,a_n)$ の計算にあたって，$f(x_1,\ldots,x_n)$ が部分ブール式として

$$\forall x g(\ldots,x,\ldots) \text{ あるいは } \exists x g(\ldots,x,\ldots)$$

を含むならばそれらは前もってそれぞれ

$$g(\ldots,0,\ldots) \wedge g(\ldots,1,\ldots),\ g(\ldots,0,\ldots) \vee g(\ldots,1,\ldots)$$

で置き換えておくものとする．すべての $a_1,\ldots,a_n \in \{0,1\}$ に対して $f(a_1,\ldots,a_n)=1$ であるとき，f は**恒真** (logically valid) であるという (ブール式のトートロジーに相当する)．f が充足不能の場合，すべての $a_1,\ldots,a_n \in \{0,1\}$ に対して $f(a_1,\ldots,a_n)=0$ であるから，f は**恒偽**であるともいう．

例 1.7 x,y,z を論理変数とする．$f=\forall x(\exists x\forall y(\neg x \wedge y) \to x)$ を考えよう．f は自由変数を含まない．

$h(x,y)=\neg x \wedge y$, $g=\exists x\forall y h(x,y)$ とおく．$h(0,1)=1$ であるから $h(x,y)$ は充足可能である．しかし，$h(1,1)=0$ であるから恒真ではない．一方，x の値によらず $h(x,0)\wedge h(x,1)=(\neg x \wedge 0)\wedge(\neg x \wedge 1)=0\wedge \neg x=0$ であるから，$x=0,1$ いずれの場合も $\forall y h(x,y)=0$，すなわち $\forall y h(0,y) \vee \forall y h(1,y)=0$ である．よって，$g=\exists x\forall y h(x,y)=0$ であり，g は充足不能 (恒偽) である．

最後に，$\exists x\forall y(\neg x \wedge y) \to \underline{x}$ を考えよう．下線を引いた x だけが自由変数であり，$\exists x\forall y(\neg x \wedge y)=0$ であるから $x=0,1$ いずれの場合もこのブール式の値は 1 である．ゆえに，f は恒真である． □

問 1.10 充足可能か恒真か恒偽か述べよ．
(a) $\forall x\forall y(x \vee y \to y)$ (b) $\exists x\forall y(x \vee y \to y)$ (c) $\forall y(x \vee y \to y)$

1.6 擬似プログラムによるアルゴリズムの記述

本書の目的は，アルゴリズムとは何かを数学的にきちんと定義し，その本質をとくに計算量の観点から追求することにある．**アルゴリズム** (algorithm) とは荒っぽい言い方をすると計算の手順のことであるが，計算がどのような計算

モデルの上で行なわれるかによってアルゴリズムの表記法は異なる．一般に，何らかの言語によりアルゴリズムを記述したものを**プログラム** (program) という．本書では**チューリングマシン** (Turing machine) と呼ばれる計算モデルを用い，一つの具体的な問題を解くことは，その問題を文字列としてコーディングしたものを入力として与えると入力を読みながら決められた動作をして最終的に問題の解を出力するようなチューリングマシンを具体的に設計することであると考える (このことは 2 章できちんと述べる)．したがって，アルゴリズムとはチューリングマシンのことに他ならず，一つの具体的チューリングマシンの動作仕様を厳密に記述したものがプログラムであるが，主としてその記述のために以下に述べる言語を用いる．このような言語を用いるのは読みやすさが旨であるから，ときに形式的な厳密さを欠く書き方をすることもあるが，必要ならいつでも厳密な定義にもとづいたものに書き直すことができる．

ここで用いる言語はプログラミング言語 Pascal によく似たものであり，制御構造以外の部分の多くは日本語で記述する．以下に述べる定義は少なからず厳密さを欠くが，そのために曖昧さや誤解が生じることはないであろう．

プログラムは，以下に述べるような**文** (statement) により構成される．

1. 代入文

変数 $\leftarrow$ 式

右辺の「式」を計算して，その値を左辺の「変数」に代入することを表わす．例えば，x の値を 1 増やすには $x \leftarrow x+1$ とする．変数や式がどのようなタイプの値をとるかは文脈から分かるので，宣言して明示することはとくにしない．

2. if 文

(2a) **if** 条件 **then** 文 **end**

(2b) **if** 条件 **then** 文 1 **else** 文 2 **end**

(2a) は，「条件」が成り立てば「文」を実行することを表わす．条件が成り立たない場合には何も行なわれない．(2b) では，条件が成り立てば「文 1」を実行し，成り立たなければ「文 2」を実行する．

3. for 文

for 変数 $\leftarrow$ 初期値 **to** 終値 **do** 文 **end**

「変数」は，ある離散全順序集合†の上を動く．変数の値を「初期値」から「終値」まで順次 1 ずつ増やしながら，「文」を繰り返し実行する．ただし，“初期値 $>$ 終値” の場合には，文は 1 回も実行されない．

† $\mathbf{N}$ の部分集合と 1 対 1 の対応をする集合を離散集合 (discrete set) という．離散全順序集合の元の間で順序を考える場合は，対応する自然数と同一視する．

変数の値を初期値から終値まで順次増やすのではなく任意の順序で，ある集合 X 上のすべての値について文を実行する場合には，次のように書く:

forall 変数 $\in X$ **do** 文 **end**

4. while 文

while 条件 **do** 文 **end**

「条件」が成り立っている間，「文」を繰返し実行する．文の実行に先立って条件が成り立つか否かのチェックが毎回行なわれる．

5. return 文

return 式

この return 文を含む関数 (あるいは手続き) の実行を終了し，この関数 (手続き) を呼び出した所に制御が戻る．関数の中に書く return 文には「式」を必ず付けなければならず，終了時における式の値が関数の値となる．手続きの中に書く return 文には式がなくてもよく，return 文に出会うと手続きの実行が終了するだけである．

6. 複合文

文 1; 文 2; $\cdots$; 文 n

「文 1」～「文 n」をひとまとめにしたものを一つの文とみなす．

7. goto 文

goto ラベル

どの文にも

ラベル : 文

のように**ラベル** (label) と呼ばれる名札を付けることができる．**goto** 文に出会うと，ラベルが付いている文へ実行の流れが移る．

8. 注釈

/* 注釈はこのように表わし，どこに置いてもよい */

さて，$x_1, \ldots, x_n$ をパラメータ (引数，argument) とする**関数**や**手続き**を定義するには次のように書く:

procedure 手続き名 $(x_1, \ldots, x_n)$
/* あるいは **function** 関数名 $(x_1, \ldots, x_n)$ */
begin
　定義記述部
end

プログラムとはこのように記述された「手続き」または「関数」のことである．

$x_1, \ldots, x_n$ は手続き (関数) の具体的内容を記述するために「定義記述部」において使われる変数であり，**仮引数** (formal parameter) と呼ばれる．定義記述部は一つの複合文である．このように定義された関数や手続きは，次の 9, 10 において「**実引数**」(actual parameter) の部分を「式 1, ..., 式 n」として呼び出すことができる．

9. 手続き呼び出し文

手続き名 (実引数)

10. 関数呼び出し

関数名 (実引数)

手続き呼び出し文はそれだけで独立した文の一つであるが，関数呼び出しは式の一部となるものであり文ではない．関数 (手続き) を呼び出すと，実引数の部分に書かれた各式の値が仮引数の中のそれぞれ対応する変数に渡されて制御がその関数 (手続き) に移る．

関数 (手続き) の定義において，定義しようとしている関数 (手続き) 自身を直接あるいは間接に呼び出してもよい．これを**再帰呼び出し** (recursive call) といい，再帰呼び出しを使って定義されている関数 (手続き) を**再帰的関数** (**再帰的手続き**) という．

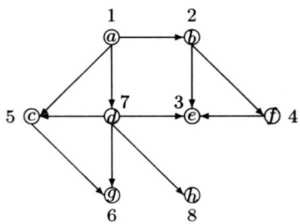


図 1.5 有向グラフの DFS

例 1.8 G を連結な有向 (あるいは無向) グラフとする．次の再帰的手続きに従って G のすべての頂点をたどる方法を**底優先探索** (**DFS**: depth-first search) という:

```
procedure DFS(x)
/* 頂点 x からスタートし，G のすべての頂点をたどる */
begin
    x をたどれ;
    forall y ∈ {y | (x, y) ∈ E(G)} do
        if y はまだたどられていない then DFS(y) end
    end
end
```

forall y の y をどのような順序で選んでゆくかは適当な基準で定める．例えば，y を選ぶ基準として「アルファベット順」を用いて，図 1.5 の有向グラフに対して DFS(a) を実行すると，図に記入した順序 (すなわち，$a-b-e-f-c-g-d-h$ の順) でたどる．すなわち，この方法では，辺に沿って可能な限り先へ先へと頂点をたどり，行き止まりになったら必要最小限だけ後戻り (バックトラック (backtrack) という) して別の可能なルートを見つけて同様にたどってゆく．

G が連結でないグラフの場合，G 全体を DFS するには連結成分 (連結な極大部分グラフのこと) それぞれに DFS をかければよい (図 1.6)． □

```
procedure overall_DFS(G)
begin
    V ← V(G);
    while V ≠ ∅ do
        x ∈ V を任意に選ぶ;  DFS(x);
        V ← V − {y | y は DFS(x) によってたどられた頂点 }
    end
end
```

図 1.6 連結とは限らないグラフ全体を DFS する

問 1.11 図 1.7 の関数は，ブール式 $f(x_1, \ldots, x_n)$ の変数 $x_1, \ldots, x_n$ に真理値 a_1, …, $a_n \in \{0, 1\}$ を代入したときの値 $f(a_1, \ldots, a_n)$ を計算する再帰的関数である．この関数を用い，QBF ϕ が充足可能か否かを判定する関数 $g(\phi)$ を作れ (ϕ が充足可能なら $g(\phi) =$ **true**，充足不能なら $g(\phi) =$ **false** とする)．

```
function f(a_1, ..., a_n)
begin
    /* f の形によって場合分けする */
    if f が論理演算記号を含まない (f = 0 または f = 1) then return f end;
    if f(x_1, ..., x_n) が ¬f_1(x_1, ..., x_n) の形 then
        if f_1(a_1, ..., a_n) = 0 then return 1 else return 0 end
    end;
    if f(x_1, ..., x_n) が f_1(x_1, ..., x_n) ∧ f_2(x_1, ..., x_n) の形 then
        if f_1(a_1, ..., a_n) = 1 かつ f_2(a_1, ..., a_n) = 1
            then return 1 else return 0
        end
    end;
    if f(x_1, ..., x_n) が f_1(x_1, ..., x_n) ∨ f_2(x_1, ..., x_n) の形 then
        if f_1(a_1, ..., a_n) = 1 または f_2(a_1, ..., a_n) = 1
            then return 1 else return 0
        end
    end;
    /* 以下，演算記号が→, ↔, ⊕の場合は省略 */
end
```

図 1.7 ブール式の値を求める関数

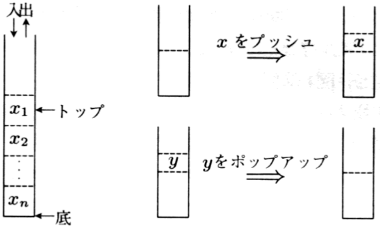


図 **1.8**　スタック

再帰的関数 (手続き) は，スタックと呼ばれるデータ構造を用いることにより再帰的ではない (**反復的** (iterative) という) 形に書き直すことができる．まず，スタックとは何かを定義しよう．ある集合の元を一列に並べた有限列

$$\langle x_1, x_2, \ldots, x_n \rangle$$

のことを**リスト** (list) という．各 x_i $(1 \leq i \leq n)$ をこのリストの**要素** (element) といい，x_1 の側あるいは x_1 自身を**トップ** (top)，x_n の側を**底** (bottom) という．要素の挿入や削除がトップにおいてしかできないようなリストのことを**スタック** (stack) という．スタックは，図 1.8 に示したように出し入れ口が 1 つしかないような容器であると考えると仕組みが理解しやすい．新しい要素 y を挿入する (**プッシュ**(push) と呼ぶ) とトップの上に積まれ，その結果 $\langle y, x_1, x_2, \ldots, x_n \rangle$ となる．取り出す (**ポップアップ** (pop up) と呼ぶ) ことのできる要素はトップだけである．例えば $\langle x_1, x_2, \ldots, x_n \rangle$ のトップをポップアップすると $\langle x_2, \ldots, x_n \rangle$ になる．

図 1.5 の有向グラフに再帰的手続き DFS(a) を適用したとき，その計算は，図 1.9 に示した根付き木 T を根を始点として底優先探索するような順序で進行する (小数字で示してある．単に「根付き木を底優先探索する」というときは，このように根を始点とし左の子を優先して進める)．T を DFS(a) の**計算木** (computation tree)，あるいは**展開** (expansion) という．T は，図 1.10 の手続き expand_DFS(x) を $x = a$ として実行すると得られる．

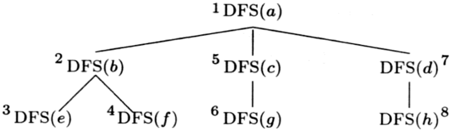


図 **1.9**　再帰的手続き DFS(a) の計算木 T

```
procedure expand_DFS(x)
begin
  "DFS(x)" をラベルとするノードを作れ (このノードを τ とする);
  forall y ∈ {y | (x, y) ∈ E(G)} do
    if y はまだたどられていない then
      expand_DFS(y) を実行してできた木を τ の子供として接ぎ木せよ
    end
  end
end
```

図 1.10 手続き DFS(x) の計算木の生成

DFS(x) は expand_DFS(x) とまったく同じ構造をしていることに注意しよう．ゆえに，DFS(x) の実行のある時点で再帰呼び出し DFS(y) が起こることと，expand_DFS(x) において DFS(y) をラベルとするノードが作られ計算木に接ぎ足されることとが 1 対 1 に対応している．また，計算木 (T とする) と，図 1.11 に示した反復的手続き iterative_DFS(x) の実行過程とは次のように 1 対 1 に対応している:

「T において，根から DFS(y) をラベルとするノードに至る道上には
ラベル DFS(y_1), ..., DFS(y_n), DFS(y) がこの順に現れる」

$$\Updownarrow$$

「iterative_DFS(x) において，y が初めてたどられた時点のスタックの
内容は，底から順に (y_1, y_2), (y_2, y_3), ..., (y_{n-1}, y_n), (y_n, y) である」

```
procedure iterative_DFS(x) /* DFS(x) の反復版 */
begin
  S を空にせよ; /* S はスタック */
  top :
    x をたどれ;
    forall y ∈ {y | (x, y) ∈ E(G)} do
      if y はまだたどられていない then
        (x, y) を S へプッシュせよ;
        x ← y;  goto top
      end;
      ℓ :
    end;
    if S が空でない then
      (x, y) の値を，S のトップをポップアップして得られた値に戻せ;
      goto ℓ
    end
end
```

図 1.11 手続き DFS(x) の反復版

procedure expand_$p(\mathbf{x})$
begin
"$p(\mathbf{x})$" をラベルとするノードを作れ (このノードを τ とする);

$p(\mathbf{x})$ の定義記述部の中の各 $p(\mathbf{y}_i)$ $(1 \le i \le n)$ を 「expand_$p(\mathbf{y}_i)$ を実行してできた木を τ の左から i 番目の子供として接ぎ木せよ」 で置き換えたもの

end

図 1.12 $p(\mathbf{x})$ の計算木の生成

最後に，DFS(x) を例にしてこれまでに述べてきたことを，一般化してまとめておこう．例えば $(x_1, \ldots, x_n)$ のような引数の組は太字 $\mathbf{x}$ で表わすことにする．

$p(\mathbf{x})$ を再帰的手続きとする．$p(\mathbf{x})$ の定義記述部は再帰呼び出し $p(\mathbf{y}_1), \ldots, p(\mathbf{y}_n)$ をこの順に含んでいるとする．図 1.12 の手続き expand_$p(\mathbf{x})$ によって作られるラベル付き根付き木を $p(\mathbf{x})$ の**計算木**という．

$p(\mathbf{x})$ を反復的手続きに直すには，再帰呼び出し $p(\mathbf{y}_i)$ の代わりに次のことを行なえばよい (図 1.13 参照)．まず，その時点における $p(\mathbf{x})$ の全状態 (いま $p(\mathbf{x})$ のどこを実行しているのか，および，すべての変数のその時点における値) をそっくりスタックにプッシュして保存してから ($p(\mathbf{x})$ の実行を一時停止して) $p(\mathbf{y}_i)$ の実行に移る．$p(\mathbf{y}_i)$ の実行が終了したら，スタックのトップをポップアップすると $p(\mathbf{y}_i)$ の実行に移る直前の $p(\mathbf{x})$ の状態 (および，実行を再開すべきプログラム内の位置) が回復されるので，実行途中であった $p(\mathbf{x})$ の実行を再開する．

問 1.12 $p(\mathbf{x})$ が再帰的関数の場合，expand_$p(\mathbf{x})$ および iterative_$p(\mathbf{x})$ をどのように修正したらよいか．

再帰的手続き $p(\mathbf{x})$，$p(\mathbf{x})$ の計算木 T，iterative_$p(\mathbf{x})$，この 3 つの間の関係を次の定理に述べる．$p(\mathbf{x})$ が再帰的関数の場合も同様なことが成り立つ．$p(\mathbf{x})$ は無限ループに入ることがないものとする．$p(\mathbf{x})$ を実行すると，時間の経過に伴って再帰呼び出し

$$(*) \quad p(\mathbf{y}_0),\ p(\mathbf{y}_1),\ \ldots,\ p(\mathbf{y}_n)$$

がこの順に起こるとする．ただし，便宜上 $p(\mathbf{y}_0) = p(\mathbf{x})$ とする．また，iterative_$p(\mathbf{x})$ の実行において，$p(\mathbf{y}_i)$ に対応する一連の動作のうち冒頭でスタックにプッシュされるものを β_i とする $(0 \le i \le n)$．ただし，β_0 は空であると定義する．

```
procedure iterative_p(x)
begin
    S を空にせよ; /* S はスタック */
    top :
    ┌─────────────────────────────────────────────
    │ p(x) の定義記述部を以下のように直したもの
    │ ただし，p(x) の末尾にも return 文があるものと考える
    │ 1. 各 p(y_i) を次の複合文で置き換える (1 ≤ i ≤ n)
    │    ┌──────────────────────────────────────
    │    │ 現時点におけるすべての変数の値および ℓ_i を S にプッシュせよ;
    │    │ /* ℓ_i は再帰呼び出しが終了したときに戻ってくる位置のラベルである */
    │    │ x ← y_i;  goto top;
    │    │ ℓ_i :
    │    └──────────────────────────────────────
    │ 2. すべての return 文を次の if 文で置き換える
    │    ┌──────────────────────────────────────
    │    │ if S が空でない then
    │    │     S のトップをポップアップせよ;
    │    │     /* ポップアップして得られた戻り位置のラベルを ℓ とする */
    │    │     すべての変数の値を，ポップアップして得られた値に戻せ;
    │    │     goto ℓ
    │    │ end
    │    └──────────────────────────────────────
    └─────────────────────────────────────────────
end
```

図 1.13　$p(\mathbf{x})$ の反復版

定理 1.5 (1) T を底優先探索したとき，出会う順にラベルを並べると (*) に等しい．

(2) 次のことが成り立つ:

「T において，根から $p(\mathbf{y}_i)$ をラベルとするノードに至る道上に
ラベル $p(\mathbf{y}_{i_1})$, ..., $p(\mathbf{y}_{i_k})$, $p(\mathbf{y}_i)$ がこの順に現れる」

$$\Updownarrow$$

「iterative_$p(\mathbf{x})$ において，$p(\mathbf{y}_i)$ に対応する複合文の
実行直前のスタック内容は底から順に β_{i_1}, ..., β_{i_k} である」

ただし，$i=0$ の場合，「$p(\mathbf{y}_i)$ に対応する複合文」とは $p(\mathbf{x})$ のことである．

証明 (1) $0 \le i \le n$ に関する帰納法で，次のことを示せばよい:

(**) T を底優先探索したとき $i+1$ 番目に出会うノードのラベルは
$p(\mathbf{y}_i)$ である．

$i=0$ のとき，1 番目に出会うノードは根であり，そのラベルは $p(\mathbf{x}) = p(\mathbf{y}_0)$ であるから (**) は成り立つ．

帰納法の仮定として，k 番目 $(0 \le k < n)$ のノードを τ としそのラベルは $p(\mathbf{y}_k)$ であるとする．底優先探索の性質より，次のことが成り立っていることに注意する:

補題 1.1 τ のどの先祖 (τ 自身を含む) においてもその左の兄弟はすべてすでにたどられている (したがって，帰納法の仮定より，ラベル $p(\mathbf{y}_0), \ldots, p(\mathbf{y}_k)$ がすでに付いている).

さて，τ に子供がある場合，底優先探索の定義より τ の次に出会うノードは τ の最も左の子であり，また，expand_$p(\mathbf{x})$ の定義より τ の最も左の子のラベルは $p(\mathbf{y}_k)$ の中で最初に起こる再帰呼び出しであり，それは $p(\mathbf{y}_k)$ の次に起こる再帰呼び出し，すなわち $p(\mathbf{y}_{k+1})$ に他ならない．τ に子供がない場合，底優先探索の定義より τ の次に出会うノードは τ のすぐ右の兄弟 (τ に右の兄弟がない場合は，右の兄弟を持つような τ の先祖のうちで τ に最も近いものを τ' とすると，τ' のすぐ右の兄弟) である．τ に子供がないということは，expand_$p(\mathbf{x})$ の定義より $p(\mathbf{y}_k)$ の中では expand_$p(\cdot)$ の再帰呼び出しが起こらないことを意味する．よって，もう一度 expand_$p(\mathbf{x})$ の定義と補題 1.1 を考慮すると，$p(\mathbf{y}_k)$ の次に起こる再帰呼び出しは τ のすぐ右の兄弟 (τ に右の兄弟がない場合は τ' のすぐ右の兄弟) のラベルとならなければならない．以上で $i = k+1$ に対しても (∗∗) が成り立つことが示された.

(2) の証明に先立って，次のことを証明しておこう.

補題 1.2 任意の $\mathbf{z}$ に対して，iterative_$p(\mathbf{z})$ の実行終了直後のスタックは空である.

証明　$p(\mathbf{z})$ の計算木 (T_z とする) のノード数 s に関する帰納法で証明する．$s = 1$ のときは明らか．$s > 1$ のとき，T_z の根の子を左から順に $\sigma_1, \ldots, \sigma_m$ とし，そのラベルをそれぞれ $p(\mathbf{z}_1), \ldots, p(\mathbf{z}_m)$ とする．計算木の定義 (expand_$p(\mathbf{z})$) より，$p(\mathbf{z}_1), \ldots, p(\mathbf{z}_m)$ は $p(\mathbf{z})$ の定義記述部に現れる再帰呼び出しである．$p(\mathbf{z}_1), \ldots, p(\mathbf{z}_m)$ の計算木は T_z の部分木でありノード数が s より少ないから，帰納法の仮定より iterative_$p(\mathbf{z}_1)$, ..., iterative_$p(\mathbf{z}_m)$ それぞれの実行終了直後のスタックは空である．iterative_$p(\mathbf{z})$ は iterative_$p(\mathbf{z}_1)$, ..., iterative_$p(\mathbf{z}_m)$ をこの順に順次模倣するだけであるから，それが終了したときのスタックもやはり空である．　□

さて，定理の (2) は i に関する帰納法で証明する．$i = 0$ のとき成り立つことは容易に確かめられる．次に，T において，根から $p(\mathbf{y}_{i+1})$ をラベルとするノード (σ とする) に至る道上のラベルを $p(\mathbf{y}_{j_1})$, ..., $p(\mathbf{y}_{j_\ell})$, $p(\mathbf{y}_{i+1})$ とする．σ の左の兄弟を $\sigma_1, \ldots, \sigma_m$ $(m \geq 0)$ とし，そのラベルをそれぞれ $p(\mathbf{y}_{k_1}), \ldots, p(\mathbf{y}_{k_m})$ とする．(1) より，$j_\ell \leq k_1 \leq \cdots \leq k_m \leq i < i+1$ である．帰納法の仮定より，iterative_$p(\mathbf{x})$ において $p(\mathbf{y}_{j_\ell})$ に対応する複合文の実行直前のスタック内容は底から順に β_{j_1}, ..., $\beta_{j_{\ell-1}}$ である．$p(\mathbf{y}_{j_\ell})$ に対応する複合文を実行することによっ

て，スタックには β_{j_ℓ} がプッシュされる．また，補題1.2により，iterative_$p(\mathbf{y}_{k_1})$，…，iterative_$p(\mathbf{y}_{k_m})$ はスタックに変化を与えないから，$p(\mathbf{y}_{i+1})$ に対応する複合文の実行直前のスタックの内容は底から順に $\beta_{j_1}, \ldots, \beta_{j_{\ell-1}}, \beta_{j_\ell}$ である．以上で $\Longrightarrow$ の証明が終わった．

$\Longleftarrow$ の証明は読者への演習問題とする． □

再帰的関数 (手続き) $p(\mathbf{x})$ の計算木 T の高さを $p(\mathbf{x})$ の**再帰の最大深さ** (maximum depth of recursion) という．

系 1.1 $p(\mathbf{x})$ のすべての変数の値を一時的に記憶しておくために必要なメモリ量を M，1つのラベル (**goto** 文の飛び先) を記憶しておくために必要なメモリ量を m とする．$p(\mathbf{x})$ の再帰の最大深さが d ならば，$p(\mathbf{x})$ を計算する反復的関数 (手続き) で，計算に必要なメモリ量が $(d+1)(M+m)$ 以下であるものが存在する． □

1.7 計算量の表わし方

実数 x に対して，x より小さくない最小の整数を $\lfloor x \rfloor$ で表わし x の**床** (floor) という．また，x を越えない最大の整数を $\lceil x \rceil$ で表わし x の**天井** (ceiling) という．

例 1.9 $\lfloor 1.2 \rfloor = \lfloor 1.0 \rfloor = \lceil 0.2 \rceil = \lceil 1.0 \rceil = 1$, $\lfloor -3.6 \rfloor = \lceil -4.7 \rceil = -4$. また，任意の実数 x に対して $\lceil x \rceil = -\lfloor -x \rfloor$. □

問 1.13 x を任意の実数，n, a, b を任意の整数 $(a, b \neq 0)$ とする．次のことを示せ．
(1) $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$.
(2) $\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor$, $\lceil \lceil n/a \rceil / b \rceil = \lceil n/ab \rceil$.

アルゴリズムが必要とする資源の量 (計算にかかる時間や必要なメモリ量など) を**計算量** (computational complexity) という．計算量は通常，入力の大きさ (= 問題のサイズ) n の関数として表わす．例えば，あるアルゴリズムの (時間に関する) 計算量が $t(n)$ であるとは，大きさが n のどんな入力に対してもこのアルゴリズムが $t(n)$ 時間以内で停止して答を出力することである．ところが，同じアルゴリズムでも，それを記述するプログラミング言語やそれを実行するコンピュータの性能によって実行時間は異なる．また，同じアルゴリズムを同じ条件下で同じ大きさの入力に対して適用しても，入力が異なれば一般に実行時間は異なる．

例 1.10 有向グラフ G と 2 頂点 $s, t \in V(G)$ が与えられたとき，s から t への道が存在するか否かを判定する問題を**グラフ到達可能性問題** (graph accessibility problem, **GAP**) という．次のの関数 GAP は，この問題を解くアルゴリズムである (道があるか/ないか (**true**/ **false**) を関数値とする)．入力は G, s, t であり，問題の大きさを表わすパラメータは G の頂点の個数 n と辺の個数 m であると考えてよいであろう．

```
function GAP(G, s, t)
begin
    1. G において DFS(s) を実行せよ;
        /*
            1.1. s をたどれ;
            1.2. forall x ∈ {x | (s, x) ∈ E(G)} do
            1.3.     if x はまだたどられていない then DFS(x) end
            1.4. end;
        */
    2. if t がたどられた then return true else return false end
end
```

行 2 の実行にかかる時間は n, m には無関係な一定時間である (c_2 とする．実行環境に依存する定数)．行 1.3 の再帰呼び出し DFS(·) は GAP(G, s, t) の全実行過程を通して高々 n 回しか起こらない．また，行 1.2 の **forall** 文の反復回数は GAP(G, s, t) の全実行過程を通して高々 $2m$ 回である (どの辺も，その両端の頂点が初めてたどられるときにそれぞれ 1 回ずつ貢献する) から，行 1.1，行 1.2〜1.3 を 1 回実行するのにかかる時間をそれぞれ c_{11}, c_{12} とすると，行 1 の実行にかかる総時間は $c_{11}n + c_{12}m$ 以下である (c_{11}, c_{12} は実行環境に依存する定数)．よって，GAP(G, s, t) の実行にかかる時間 $t(n)$ は最悪の場合 $c_2 + c_{11}n + c_{12}m$ である．ただし，s, t によってはこれよりずっと早い時間で終了する場合もある．

頂点数だけに注目する場合には，$m \le n^2$ であるから，実行時間は (大雑把ではあるが) n^2 に比例する時間以下であるということができる． □

このように，関数値の上界だけを問題にしたい場合とか，実行環境によって定数倍の差が生じるような関数すべてを本質的には同じものであるとみなしたい場合などに，次のような記法が便利である*.

f, g を $\mathbf{N}$ から $\mathbf{N}$ への関数とする．ある定数 $c > 0$ と自然数 n_0 が存在して，

---

*O 記法は P.Bachmann により (1892 年)，o 記法は E.Landau により導入され (このため，O 記法や o 記法はランダウの記法とも呼ばれる) 昔から使われていたが，Ω, ω, Θ などの記法は D.E.Knuth によって導入された記法である: D.E.Knuth, Big omicron and big omega and big theta, *ACM SIGACT News* 8, pp.18-24, 1976.

$n \ge n_0$ なる任意の自然数 n に対して $f(n) \le cg(n)$ が成り立つとき，$f(n)$ の**オーダー** (order) は $g(n)$ であるといい，

$$f(n) = O(g(n))$$

と表わす．このとき，$g(n)$ を $f(n)$ の**漸近上界** (asymptotic upper bound) という．これは荒っぽく言うと，"n が十分大きければ" $f(n)$ は $g(n)$ の定数倍以下である，ということである．

O 記法と対をなすのが Ω 記法である．ある定数 $c > 0$ と自然数 n_0 が存在して，$n \ge n_0$ なる任意の自然数 n に対して $f(n) \ge cg(n)$ が成り立つとき

$$f(n) = \Omega(g(n))$$

と表わし，$g(n)$ を $f(n)$ の**漸近下界** (asymptotic lower bound) という．$f(n) = O(g(n))$ かつ $f(n) = \Omega(g(n))$ であるとき

$$f(n) = \Theta(g(n))$$

と表わし，$f(n)$ と $g(n)$ は**同じオーダー**であるという．

例 1.11 (1) $O(1)$ は定数を表わす．

(2) $n \ge 3$ なら $5n \le 5n+3 \le 6n$ であるから，$5n+3 = \Theta(n)$ である．一般に，自然数を係数とする k 次多項式 $p(n) = a_k n^k + \cdots + a_1 n + a_0$ $(a_k \ne 0)$ に対して，$p(n) = \Theta(n^k)$ である．

(3) n が偶数のとき $f(n) = n$, n が奇数のとき $f(n) = \log n$ と定義すると，$f(n) = O(n)$，$f(n) \ne O(\log n)$, $f(n) = \Omega(\log n)$，$f(n) \ne \Omega(n)$ である．　□

問 1.14 どんな実数 $c > 0$ に対しても十分大きな自然数 n を選べば $cf(n) < g(n)$ とできるとき，$f(n)$ の**オーダー**は $g(n)$ より**真に低い**といい，$f(n) = o(g(n))$ で表わす．次のことを証明せよ．

(a) $f(n) = o(g(n)) \Longleftrightarrow \lim_{n\to\infty} f(n)/g(n) = 0$.

(b) 任意の実数 $\epsilon > 0$ に対して $\log n = o(n^\epsilon)$.

O 記法等は $n^2 + 3n + 4 = n^2 + O(n)$, $n + \log n = n + o(n)$ のように式の中で用いることもある．一般に，例えば $f(n) = O(g(n))$ のとき，$f(n) + g(n) = O(g(n)) + g(n) = O(g(n))$ のように書く．

例 1.12 (1) $f(n)$ が n のベキであるとき，$f(n) = n^{O(1)}$ と書く．また，ある定数 $c > 0$ が存在して無限個の n に対して $g(n) \ge n^c$ であるとき，$g(n) = n^{\Omega(1)}$ と書く．

(2) $f(n) = 2^{O(g(n))}$ は，ある定数 $c > 0$ と n_0 が存在して $n \ge n_0$ なる任意の自然数 n に対して $f(n) \le 2^{cg(n)}$ が成り立つことを表わす．

(3) $f(0) = 1$，$f(n) = n + \frac{1}{2} f(\lfloor \frac{2}{3} n \rfloor)$ のとき，問 1.13 を考慮すると，

$$
\begin{aligned}
f(n) &\le n(1+1/3+1/3^2+\cdots+1/3^k) + f(\lfloor(\tfrac{2}{3})^{k+1}n\rfloor)/2^{k+1} \\
&\le n(1+1/3+1/3^2+\cdots+1/3^{\lceil \log_{3/2} n\rceil}) + f(0)/2^{\log_{3/2} n} \\
&\le n\sum_{i=0}^{\infty}(1/3)^i + n^{\frac{1}{\log_2 3-1}} \\
&\le \tfrac{3}{2}n + o(n) = \Theta(n) + o(n) = \Theta(n). \quad \Box
\end{aligned}
$$

$h(n)$ を $\mathbf{N}$ から実数への関数とする．$\{h(n), h(n+1), \ldots\}$ の上限の $n \to \infty$ としたときの極限値 ($\pm\infty$ の場合も含む) を $h(n)$ の**上極限** (limes superior) といい，$\displaystyle\sup_{n\to\infty} h(n)$ で表わす．同様に，下限の極限値を $\displaystyle\inf_{n\to\infty} h(n)$ で表わし，**下極限** (limes inferior) という．例えば，$h(n) = (-1)^n + \dfrac{1}{n+1}$ とすると，$\{h(n), h(n+1), \ldots\}$ の上限は n が偶数のとき $1 + \dfrac{1}{n+1}$，n が奇数のとき $1 + \dfrac{1}{n+2}$ であり，下限は -1 であるから，$\displaystyle\sup_{n\to\infty} h(n) = 1$，$\displaystyle\inf_{n\to\infty} h(n) = -1$ である．

問 1.15 (1) オーダーの違いを述べよ．
(a) $(2n)^2$ (b) 2^{2n} (c) 2^{2^n} (d) 2^{n^2} (e) $(2^n)^2$ (f) $(n^2)^2$
(2) 次の $f(n)$ について，上極限と下極限，および，$f(n) = O(g(n))$，$f(n) = \Omega(h(n))$ を満たす最良の $g(n)$，$h(n)$ を求めよ．
(a) n が偶数のとき $f(n) = n$，奇数のとき $f(n) = 1/n$.
(b) $f(n) = n^{\sin n}$.

定理 1.6 f と g を $\mathbf{N}$ から $\mathbf{N}$ への関数とすると，次のことが成り立つ．
(1) $f(n) = O(g(n)) \iff \displaystyle\sup_{n\to\infty} f(n)/g(n) < \infty$.
(2) $f(n) = \Omega(g(n)) \iff \displaystyle\inf_{n\to\infty} f(n)/g(n) < \infty$.

証明 (1) $h(n) = f(n)/g(n)$ とおき，$\{h(n), h(n+1), \ldots\}$ の上限を $\overline{h(n)}$ とおくと，$\overline{h(n)}$ は単調減少である．

$f(n) = O(g(n))$ とすると，ある定数 $c > 0$ と自然数 n_0 が存在して，n_0 以上の任意の n に対して $h(n) \le c$ である．したがって，$n \ge n_0$ なる任意の n に対して $\overline{h(n)} \le c$ である．ゆえに，$\displaystyle\lim_{n\to\infty} \overline{h(n)} \le c$ であり，これで $\displaystyle\sup_{n\to\infty} h(n) \le c < \infty$ が示された．

逆に，$\displaystyle\sup_{n\to\infty} h(n) = c < \infty$ とすると $\displaystyle\lim_{n\to\infty} \overline{h(n)} = c$ で $h(n) \le \overline{h(n)}$ であるから，$\displaystyle\lim_{n\to\infty} h(n) \le c$ である．したがって，どんな $\epsilon > 0$ に対しても十分大きい n_0 をとれば，n_0 以上の任意の n に対して $h(n) \le c + \epsilon$ が成り立つ．よって，$f(n) = O(g(n))$ である．

(2) も (1) と同様に証明できる. □

問 1.16 (1) 定理 1.6 の (2) を証明せよ.
(2) $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$ であることを証明せよ.

演習問題 1

1 アルファベット $\Sigma = \{a, b, c\}$ 上の語の上の関係 $\sqsupset, \sqsupseteq$ を,

$$x \sqsupseteq y \overset{\text{def}}{\iff} x = zy \text{ なる } z \in \{a,b\}^* \text{ が存在する}$$
$$x \sqsupset y \overset{\text{def}}{\iff} x \sqsupseteq y \text{ かつ } x \neq y$$

で定義する.

(a) $w = abacba$ とするとき,

$$W_1 = \{w' \mid w \sqsupset w'\},\ W_2 = \{w' \mid w \sqsupset^2 w'\},\ W_\infty = \{w' \mid w \sqsupset^* w'\}$$

をそれぞれ求めよ.

(b) $\sqsupseteq$ は半順序であることを示せ.

2 $i = 1, 2, \ldots$ に対して, **Z** 上の 2 項関係 R_i を次のように定義する:

$$xR_i\,y \overset{\text{def}}{\iff} |\,x - y\,| = i.$$

(a) $xR_i^k\,y$ となるための x, y の関係を k で表わせ.
(b) $R_i^+ \cap R_j^+ = \emptyset$ となるための条件を求めよ.
(c) $R_i \circ R_j$ を求めよ.

3 次の 3 つの条件

(i) 任意の $a \in X$ に対して aRa (反射律)
(ii) 任意の $a, b \in X$ に対して $aRb \Longrightarrow bRa$ (対称律)
(iii) 任意の $a, b, c \in X$ に対して $aRb, bRc \Longrightarrow aRc$ (推移律)

を満たす関係 R を X 上の**同値関係** (equivalence relation) といい,

$$[a]_R = \{b \in X \mid aRb\}$$

を a を**代表元** (representative) とする**同値類** (equivalence class) という.

(1) 整数 x, y, m に対し, m が $x - y$ を割り切るとき x と y は m **を法として合同**であるといい,

$$x \equiv y \pmod{m}$$

と書く. m を任意に固定したとき,「m を法として合同である」という関係は同値関係であることを証明し, $m = 3$ のときの同値類を示せ.

(2) $x \equiv x' \pmod{m}$, $y \equiv y' \pmod{m}$ ならば $x + y \equiv x' + y' \pmod{m}$ であることを示せ.

(3) a, b を整数とする. x を未知数とする合同式 $ax \equiv b \pmod{m}$ が解を持つ必要十分条件は, a, m の最大公約数が b を割り切ることである. このことを証明せよ.

3 有向グラフ $G = (V, E)$ において, E を V 上の 2 項関係とみたとき, 次のことをグラフの用語で答えよ.

(a) E が同値関係になるための条件とそのときの同値類
(b) E が半順序となるための条件，全順序になるための条件

4 T は n 個のノードからなり，m 個の葉を持つ 2 分木とする．

$$\mathrm{height}(T) \geq \log_2(n+1) - 1,\ \mathrm{height}(T) \geq \log_2 m$$

であることを示せ．

5 $\Sigma = \{\langle, \rangle\}$ とし，D を，問 1.3 で定義した言語とする．Σ^* の任意の語 x, y に対して，関係 $\sim$ を $x\langle\rangle y \sim xy$ で定義する．$\sim^*$ が同値関係であること，および，$D = [\lambda]_{\sim^*} = \{x \in \Sigma^* \mid x \sim^* \varepsilon\}$ であることを示せ．

6 x, y, z を論理変数，α, β, γ を任意のブール式とする．充足可能か，恒真か，恒偽かを述べよ．
(a) $(x \to y) \to \forall z(x \wedge z \to y \wedge z)$ (b) $\exists x \forall y \alpha \to \forall y \exists x \alpha$
(c) $\forall x(\alpha \wedge \beta) \to (\forall \alpha \wedge \forall \beta)$ (d) $\exists x(\alpha \vee \beta) \to (\exists \alpha \vee \exists \beta)$

7 **幅優先探索** (**BFS**: breadth-first search) と呼ばれる方法は DFS と違い，たどることを先へ先へと進めるのではなく，v から出ている各辺の端点がそれぞれ $u_1, u_2, \ldots, u_n$ であったら v の次には $u_1, u_1, \ldots, u_n$ の順にたどり，その次に u_1 から出ている各辺の端点であるすべての頂点，u_2 から出ている各辺の端点であるすべての頂点，... という順序でたどる．例 1.5 のグラフを BFS でたどれ．

8 (1) $\log n! = \Theta(n \log n)$ であることを示せ．
(2) $f(0) = 0,\ f(n) = f(n/3) + f(2n/3) + n$ ならば，$f(n) = O(n \log n)$ であることを示せ．
(3) どんな実数 $c > 0$ に対しても十分大きな自然数 n を選べば $f(n) > cg(n)$ とできるとき $f(n)$ の**オーダーは** $g(n)$ より**真に高い**といい，$f(n) = \omega(g(n))$ で表わす．$f(n) = o(g(n)) \Longleftrightarrow g(n) = \omega(f(n))$ であることを証明せよ．

9 全順序集合 $A = \{a_1, a_2, \ldots, a_n\}$ が与えられたとき，その元を大きさの順 (**昇順** (ascending order): $a_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_n}$，または**降順** (descending order): $a_{j_1} \geq a_{j_2} \geq \ldots \geq a_{j_n}$) に並べ替えることを**ソーティング** (sorting) という．n 個の元を 2 数の大小関係 ($a_i \geq a_j$?) を比べるだけでソートするためには，最悪の場合 $\Omega(n \log n)$ 回の比較が必要であることを示せ．ヒント: 2 数の比較結果で左右に分岐し，葉にはソートされた結果がくるような 2 分木を考えよ．どんなアルゴリズムに対してもこの木の根から葉への道が 1 つ対応する．葉には $a_1, a_2, \ldots, a_n$ の $n!$ 通りの順列すべてがなければならない．木の高さは最悪の場合の比較回数である．

10 2 つのプログラム A と B がある．入力データの個数が n のとき，A, B それぞれの実行時間 $a(n), b(n)$ は

$$\begin{cases} a(2n) = 2a(n) + n^2 \\ b(3n) = b(2n) + b(n) + n \end{cases} \quad (n \geq 1)$$

という性質がある．A, B どちらのプログラムの方が効率がよいか？

2
チューリングマシン

第1章で「アルゴリズムとは計算の手順のことであり，アルゴリズムが必要とする資源の量 (計算にかかる時間や必要なメモリ量など) を計算量という」と定義した．ただ素朴にアルゴリズムを示してみせるだけであれば，この定義で何ら困ることはない．しかし，2つのアルゴリズムの計算量の比較を精細にやりたいのであれば，アルゴリズムを実行する媒体である計算機モデルを明確にする必要がある．さらに，ある問題についてそれを解くためのアルゴリズムが存在しないことを証明するためには，そもそもアルゴリズムとは何かをきちんと定義しなければならない．この章では，アルゴリズムと計算量の厳密な定義をし，その基本的な性質について考察する．

2.1 アルゴリズムとは (歴史的背景)

普通われわれが桁数の大きい2数の足し算を行なうときには，10進表記して下の桁から1桁ずつ順に桁上がりを考慮しながら足していく．この操作手順は値という概念からは離れて行なわれる機械的なものである．このようなまったく機械的な計算・操作の手順 (手続き) のことをいつの頃からかアルゴリズムと呼ぶようになった．アルゴリズムという名称は，9世紀のアラビアの数学者 Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî の名前に由来する．ちなみに，彼は *Kitab al jabr w'al-muqabala* という著書でアラビア式10進記法について記しており，この書名から algebra (代数) という言葉が生まれた．しかし，今世紀の中頃まではアルゴリズムと言えば数の四則算法，あるいは「ユークリッドのアルゴリズム」(2つの自然数の最大公約数を求めるための方法である**ユークリッドの互除法**のこと) を指すのが普通で，英語の綴りも今日用いられている algorithm ではなく algorism であった．アルゴリズムという概念の必

要性が学問的にも実際的にも当時はそれほどなく，アルゴリズムとは何かという数学的な定義などなくても困ることはなかった．どんな問題に対してもそれを解くアルゴリズムは必ずあるものだと誰もが信じていたし，実際，ほとんどすべての問題はアルゴリズムを示すことができたからである．

アルゴリズムとは何かを数学的にきちんと定義する必要を人々が認識するようになったのは 1930 年台になってからである．その発端は今世紀の大数学者の一人ヒルベルトが 1900 年にパリで開催された国際数学者会議において行なった講演にある*．その中で彼は当時の数学が解決すべき重要な問題をいくつか提示したが，その 10 番目の問題は次のようなものであった:

ヒルベルトの第 10 問題 (Hilbert's tenth problem)

$P(x_1,\ldots,x_n)$ を，整数を係数とする $x_1,\ldots,x_n$ の多項式とするとき，$P(x_1,\ldots,x_n)=0$ が自明 (すべてが 0) でない整数解をもつかどうかを判定するアルゴリズムを求めよ．

実は，ヒルベルトは数学の体系を完全に形式化・公理化して，数学のすべての定理を機械的に証明することを構想していた (**ヒルベルトのプログラム**)．数学の定理や証明を記号化して形式的に扱う論理体系として 1 階述語論理 (first-order theory) と呼ばれるものがある．ヒルベルトの構想では，1 階述語論理において，ある命題を表わす論理式が公理から導かれる (定理として証明できる) かどうかを判定するアルゴリズムを求めることがキーポイントであった (ヒルベルトはこの問題を**決定問題** (Entscheidungsproblem) と呼んだ)．第 10 問題は，決定問題の非常に特別な場合である．

多くの人々が第 10 問題を解くアルゴリズムを求めるために努力したが徒労に終わった．なぜなら，そのようなアルゴリズムは実は存在しなかったからである (そのことはずっと後の 1970 年に Ju.v.Matijasevič によって証明された†)．そんな折，ヒルベルトの弟子のゲーデルが衝撃的な定理を証明した (**不完全性定理**: Incompleteness Theorem for S) ‡．それは，自然数の体系を公理化した 1 階述語論理を S とするとき，S が無矛盾 (ある命題とその否定が同時に S の定理となることはない) ならば S の中で成り立つことも成り立たないことも示せないような命題が存在する，というものである．これにより，**決定不能** (undecidable，アルゴリズムが存在しない) 問題の存在が初めて示され，ヒルベ

---

*D.Hilbert, Mathematical problems, *Bull. Amer. Math. Soc.* 8, pp.437-445, 478-479, 1901.

†Ju.v.Matijasevič, Diofantovo predstavlenie perechislimykh predikatov, *Izvestija Akademii Nauk SSSR, Ser. Matem.* 35, pp.3-30, 1971.

‡K.Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, *Monatshefte für Mathematik und Physik* 38, pp.173-198, 1931.

ルトの構想は挫折した．しかし，第 10 問題が否定的に解決されたわけではない．とはいえ，第 10 問題に対するアルゴリズムは存在しないのではないかと人々が疑い始めるきっかけになったことは間違いあるまい．

アルゴリズムが存在しないことを証明するためには，アルゴリズムとは何かを厳密に定義する必要がある．ゲーデルの結果以後，アルゴリズムとは何かを数学的に厳密に定義する試みが数多くなされ，1930 年代はアルゴリズム研究の黎明期となった．それらは

- 1934 年 K.Gödel & J.Herbrand: Herbrand-Gödel 計算可能関数 (HG-computable function)
- 1936 年 A.M.Turing: チューリングマシン (Turing machine)
- 1936 年 S.C.Kleene: 帰納的関数 (recursive function)
- 1936 (1941) 年 A.Church: λ定義可能関数 (λ-definable function)
- 1936 (1943) 年 E.L.Post: ポスト正規システム (Post canonical system)

であり，それより後に定義された

- 1954 年 A.A.Markov: マルコフアルゴリズム (Markov normal algorithm)
- 1956 年 N.Chomsky: 句構造文法 (phrase-structure grammar)
- 1963 年 J.C.Shepherdson & H.E.Sturgis: ランダムアクセスマシン (RAM, random access machine)
- 1964 年 C.C.Elgot & A.Robinson: プログラム内蔵方式 RAM (random access stored program machine)
- 1967 年 A.R.Meyer & D.M.Ritchie: while プログラム (while program)

も含め，これらすべてが「計算可能性」，すなわち「計算を行なうためのアルゴリズムを有す」という概念を定義する等価なものであることが明らかになった．1936 年，チャーチが

チャーチの提唱 (Church's thesis)
「計算可能関数とは帰納的関数のことに他ならない (換言すると，チューリングマシンによって記述できるものをアルゴリズムと呼ぼう)」

と提唱し[§]，それ以後，これがアルゴリズムの定義として用いられることになり，今日では数学者の間でほとんど確定した見解となっている．

本節では，while プログラム，帰納的関数について概略を述べる．これらは本書の主題と直接の関係はないが，現在もコンピュータサイエンスの各分野で

[§] A.Church, An unsolvable problem of elementary number theory, *Amer. J. Math.* 58, pp.345-363, 1936.

それぞれ非常に重要な役割を果している．チューリングマシンは本書の主題であるから節を改めて定義し (2.2 節)，2.5 節で句構造文法との関係について述べる．ランダムアクセスマシンについては，後ほど並列計算の章 (6.3 節) で取り上げる．λ定義可能関数，Herbran-Gödel 計算可能関数他の概念は演習問題 2 の中で触れる．

2.1.1　while プログラム

現代人の感覚では，コンピュータを使って計算を行なうための手順をプログラムとして書き表わすことができるような関数こそ計算可能な関数と考えるのではないだろうか．そこでここでは，もっとも原始的な「プログラム」をきちんと定義しよう．それが while プログラムであり，以下のように再帰的に定義される¶．以下の定義式において <> で括られたものは，while プログラムを定義するために補助的に導入される各種のプログラム要素を表わし，::= は，::= の左辺のプログラム要素が右辺のような形をした文字列として定義されることを表わす．右辺に現れる記号 | は「または」を表わす．このような記法を **BNF** (Backus-Naur form) という．

<代入文> ::= <変数>←0 | <変数>←<変数> | <変数>←<変数>+1
<loop文> ::= **loop** <変数> **do** <文> **end**
<while文> ::= **while** <変数> **do** <文> **end**
<文> ::= <代入文> | <loop文> | <while文> | <文> ; <文>
<whileプログラム> ::=
　　<関数名> (<引数>) → <出力変数> **begin** <文> **end**
<引数> ::= <入力変数> | <引数> , <入力変数>

以下の while プログラムの例において，何が変数であるかは，文脈から容易に読み取れるので明記していない．すべての変数は自然数だけを値にとる．

x を変数，S を文とする．**loop** x **do** S **end** に出会う直前の x の値が n であるとき，loop 文

$$\textbf{loop}\ x\ \textbf{do}\ S\ \textbf{end}$$

は S を n 回反復実行することを表わす (たとえ S の中で x の値が変わっても反復回数は変わらない)．また，while 文

$$\textbf{while}\ x\ \textbf{do}\ S\ \textbf{end}$$

は $x > 0$ である限り S を反復実行することを表わす (最初から $x = 0$ なら S

¶A.R.Meyer and D.M.Ritchie, The complexity of loop programs, *Proc. ACM Natl. Conf.*, pp.465-469, 1967.　D.M.Ritchie は C 言語の設計者として有名なその人である．

は 1 回も実行されない．また，$x > 0$ のまま x の値が S の中で変化しなければ無限ループに陥る)．while プログラム

$$P(x_1, \ldots, x_n) \to y$$
$$\textbf{begin } S \textbf{ end}$$

の入力変数 $x_1, \ldots, x_n$ に初期値 $a_1, \ldots, a_n$ を与えて P の実行を開始し，末尾の **end** に至って実行を終了したときの出力変数 y の値が b であるならば，$P(a_1, \ldots, a_n) = b$ と書くことにする．ループに陥ってしまい実行が終了しないときには $P(a_1, \ldots, a_n)$ は定義されない．P は $\mathbf{N}^n$ から $\mathbf{N}$ への関数 (部分関数の場合もある)

$$P : (a_1, \ldots, a_n) \mapsto b$$

とみることができる．このように while プログラムによって定義できる関数を **while 関数**と呼ぶ．

例 2.1 x, y, u, v を変数とし，subt を関数名とする．

```
subt(x, y) → x
begin
    loop y do
        u ← 0; v ← 0;
        loop x do v ← u; u ← u + 1 end;
        x  ← v
    end
end
```

において，$v \leftarrow u$ と $u \leftarrow u + 1$ はともに代入文であるから文でもある．したがって，それらを ; で結んだ $v \leftarrow u;\ u \leftarrow u + 1$ も文であり，したがって，

$$\textbf{loop } x \textbf{ do } v \leftarrow u;\ u \leftarrow u + 1 \textbf{ end}$$

も文 (**loop** 文) である．文

```
u ← 0; v ← 0;
loop x do v ← u; u ← u + 1 end;
x  ← v
```

を実行すると x の値は (実行前の x の値) $- 1$ となる．**loop** y **do** $\cdots$ **end** によってこれを y 回繰り返すと，**begin** $\cdots$ **end** の実行が終了したときの x の値は (実行前の x の値) $- y$ となる．すなわち，この while プログラムは関数

$$\mathrm{subt}(x, y) = \begin{cases} x - y & x \geq y \text{ のとき} \\ 0 & x < y \text{ のとき} \end{cases}$$

を定義している． □

$f(x_1, \ldots, x_n) \to y$ **begin** P **end** を while プログラムとし，Q を文とする．Q において

$$z \leftarrow f(a_1, \ldots, a_n)$$

と書くのは，

$$\begin{aligned}&\bar{x}_1 \leftarrow a_1;\ \cdots;\ \bar{x}_n \leftarrow a_n;\\&\bar{P};\\&z \leftarrow \bar{y}\end{aligned}$$

の省略表記である．ただし，$\bar{P}$ は，P の中に現れるすべての変数 (その一つを例えば x とする) を，Q に現れない別の変数で (例えば x を $\bar{x}$ で) 置き換えたものである．例えば，

```
f(x) → x
begin
    ┌─────────────────────────────────────┐
    │ u ← 0; v ← 0;                        │
    │ loop x do v ← u; u ← u + 1 end;      │
    │ x  ← v                               │
    └─────────────────────────────────────┘
end
```

のとき，

```
g (x, y) → x
begin
    loop y do x ← [f(x)] end
end
```

は

```
g (x, y) → x
begin
    loop y do
        x̄ ← x;
        ┌─────────────────────────────────────┐
        │ ū ← 0; v̄ ← 0;                        │
        │ loop x̄ do v̄ ← ū; ū ← ū + 1 end;      │
        │ x̄  ← v̄                               │
        └─────────────────────────────────────┘
        x ← x̄
    end
end
```

の略記であり，$g(x, y)$ は $\mathrm{subt}(x, y)$ に等しい．

問 2.1 上記のような略記法を用いて，次の関数を定義する while プログラムを順次作れ．

(a) $\mathrm{mult}(x, y) = xy$
(b) $\mathrm{div}(x, y) = x$ を y で割った商 (ただし，$y = 0$ のときは 0)
(c) $\mathrm{sqrt}(x) = \lfloor\sqrt{x}\rfloor$

2.1.2 帰納的関数

帰納的関数を，以下の (1) ～ (6) によって再帰的に定義する[‖]．まず，自然数の上で定義される次の 3 種の関数 (1) ～ (3) を**初期関数** (initial function) と呼ぶ．初期関数は帰納的関数である．

(1) zero : $\mathbf{N}^0 \to \mathbf{N}$ **零関数** (zero function)

zero は 0 変数関数で，その値はつねに 0 である．すなわち，$\mathrm{zero}() = 0$.

(2) succ : $\mathbf{N} \to \mathbf{N}$ **後者関数** (successor)

succ は自然数 x に「x の次の自然数」$x+1$ を対応させる関数である．すなわち，任意の $x \in \mathbf{N}$ に対して $\mathrm{succ}(x) = x+1$.

(3) $\pi_i^n : \mathbf{N}^n \to \mathbf{N}$ **射影** (projection)

π_i^n は n 個の組 $(x_1, \ldots, x_n)$ の i 番目の成分 x_i を取り出す関数である．すなわち，$\pi_i^n(x_1, \ldots, x_n) = x_i$. ただし，$n \in \mathbf{N}$, $0 \le i \le n$ で，とくに $\pi_0^n(x_1, \ldots, x_n)$ は空を表わす (例えば，$\mathrm{zero}(\pi_0^n(x_1, \ldots, x_n)) = \mathrm{zero}()$ である)．

(4) $f : \mathbf{N}^m \to \mathbf{N}$ および $g_i : \mathbf{N}^n \to \mathbf{N}$ $(i = 1, \ldots, m)$ がそれぞれ帰納的関数 (部分関数の場合を含む) であるならば，

$$f(g_1, \ldots, g_m)(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$$

によって定義される**合成関数** $f(g_1, \ldots, g_m) : \mathbf{N}^n \to \mathbf{N}$ も帰納的 (部分) 関数である．

(5) $g : \mathbf{N}^n \to \mathbf{N}$ と $h : \mathbf{N}^{n+2} \to \mathbf{N}$ がそれぞれ帰納的関数 (部分関数の場合を含む) であるならば $(n \ge 0)$,

$$\begin{cases} f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n) \\ f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \end{cases}$$

によって定まる (部分) 関数 $f : \mathbf{N}^{n+1} \to \mathbf{N}$ も帰納的 (部分) 関数である．このような関数の定義法を**原始帰納法** (primitive recursion) という．

(6) $g : \mathbf{N}^{n+1} \to \mathbf{N}$ が帰納的関数 (部分関数の場合を含む) であるならば，

$$f(x_1, \ldots, x_n) = \text{「}g(x_1, \ldots, x_n, y) = 0 \text{ となる最小の自然数 } y\text{」}$$

によって定まる (部分) 関数 $f : \mathbf{N}^n \to \mathbf{N}$ も帰納的 (部分) 関数である．このような関数の定義法を**最小化** (minimization) という．

(1) ～ (6) で定まる (部分) 関数だけを**帰納的 (部分) 関数** ((partial) recursive function) という．

例 2.2 (1) one = succ(zero), two = succ(one), ... によって定まる one, two, ... : $\mathbf{N}^0 \to \mathbf{N}$ はそれぞれ定数 $1, 2, \ldots$ を表わす帰納的関数である．

[‖]S.C.Kleene, General recursive functions of natural numbers, *Math. Ann.* 112, pp.727-742, 1936.

(2) 2 数の和を与える関数 $\mathrm{add}(x,y) = x + y$ は

$$\begin{cases} \mathrm{add}(x,0) = \pi_1^1(x) \\ \mathrm{add}(x,y+1) = \mathrm{succ}(\pi_3^3)(x,y,\mathrm{add}(x,y)) \end{cases}$$

と定義できるので，帰納的関数である．これは

$$\begin{cases} x + 0 = x \\ x + (y+1) = (x+y) + 1 \end{cases}$$

を上記の定義通りにきちんと表現したものに他ならない．同様に，

$$\begin{cases} x \cdot 0 = 0 \\ x \cdot (y+1) = (x \cdot y) + x \end{cases} \quad \begin{cases} \mathrm{mult}(x,0) = \mathrm{zero}(\pi_0^1)(x) \\ \mathrm{mult}(x,y+1) = \mathrm{add}(\pi_3^3,\pi_1^3)(x,y,\mathrm{mult}(x,y)) \end{cases}$$

であるから，$\mathrm{mult}(x,y) = x \cdot y$ も帰納的関数である．この定義に従って $\mathrm{mult}(2,2)$ がどのように "計算" されるかを見てみよう：

$$\begin{aligned}
\mathrm{mult}(2,2) &= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{mult}(2,1)) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{add}(\pi_3^3,\pi_1^3)(2,0,\mathrm{mult}(2,0))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{add}(\pi_3^3,\pi_1^3)(2,0,\mathrm{zero}(\pi_0^1)(2))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{add}(\pi_3^3,\pi_1^3)(2,0,\mathrm{zero}(\pi_0^1(2)))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{add}(\pi_3^3,\pi_1^3)(2,0,\mathrm{zero}())) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{add}(\pi_3^3,\pi_1^3)(2,0,0)) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{add}(0,2)) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{succ}(\pi_3^3)(0,1,\mathrm{add}(0,1))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{succ}(\pi_3^3)(0,1,\mathrm{succ}(\pi_3^3)(0,0,\mathrm{add}(0,0)))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{succ}(\pi_3^3)(0,1,\mathrm{succ}(\pi_3^3)(0,0,\pi_1^1(0)))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{succ}(\pi_3^3)(0,1,\mathrm{succ}(\pi_3^3)(0,0,0))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{succ}(\pi_3^3)(0,1,\mathrm{succ}(0))) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{succ}(\pi_3^3)(0,1,1)) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,\mathrm{succ}(1)) \\
&= \mathrm{add}(\pi_3^3,\pi_1^3)(2,1,2) \\
&= \mathrm{add}(2,2) = \cdots = 4. \quad \square
\end{aligned}$$

問 2.2 上記の (6) を使わないで定義される関数を**原始帰納的関数** (primitive recursive function) という．したがって，例 2.2 の関数はいずれも原始帰納的関数である．

(1) 自然数 x に対して定義された次の関数は原始帰納的関数であることを示せ:

(a) $\max(x,y)$　(b) $\mathrm{fact}(x) = x!$　(c) $\mathrm{sqrt}(x) = \lfloor\sqrt{x}\rfloor$

(2) 原始帰納的ではない帰納的部分関数の例を示せ．ヒント: 原始帰納的関数は全域関数である．演習問題 2 も参照せよ．

命題 2.1 自然数の上の (部分) 関数 f が帰納的 (部分) 関数であることと，f が while 関数であることとは同値である．

2.2 多テープチューリングマシン

前節でさまざまな計算モデルについて述べた．現在でもそれらはコンピュータサイエンスのさまざまな領域でそれぞれ重要な役割を果たしているが，本書の主題である計算量の理論においては，計算のモデルとして通常用いられるのはチューリングマシンと呼ばれるものである**．チューリングマシンが用いられる理由は，モデルとして非常に単純で取り扱いが容易であること，それでいながら高い能力を持つこと，そして最大の理由は，計算量理論の歴史の初期から今日に至るまで標準的な計算モデルとして広く使われてきたために理論的に安定していること，等である．チューリングマシンの基本的構成要素である「テープ」は入出力装置と記憶装置を兼ねるもので，チューリングマシンは一見して，磁気テープ装置を何台も持つコンピュータとよく似ている．しかし，チューリング (Alan M.Turing) がこれを考え出した1930年代には，まだコンピュータが発明されていなかったことは注目すべき事実である．

以後，チューリングマシンを **TM** と略す．本節では**多テープ TM** (multitape Turing machine) について述べる．

テープ (tape) とは，図 2.1 に示したように，**ます目** (cell, square) が右方向に無限につづいたものである．ます目には，左端から順に 1, 2, ... と番号が付いているものとする．各ます目には1つの記号 (文字) を書くことができる．各テープにはヘッド (head) が1つ付いていて，このヘッドによって文字を読んだり書き換えたりする (第 i テープのヘッドをヘッド i と呼ぶことにする)．多テープ TM は (有限) 複数個のテープを持つことができ，そのうちの1本を**入力テープ** (input tape) として使う．それ以外のテープを**作業用テープ** (work tape) という．テープ (入力テープを含む) 上に書くことができる記号をテープ記号という．何も書かれていないテープのます目には，空白記号と呼ばれる特別の記号 ($\flat$ で表わす) が書かれているとみなす．**有限制御部** (finite control) には有限量の情報を記憶することができる．有限制御部に記憶された情報を「状態」と呼ぶ．

TM は，(1) 各ヘッドが見ている文字を読み取ること，(2) 各ヘッドが見ているます目に文字を書き込むこと，(3) 各ヘッドを右または左に1こま移動させるか，同じ位置にとどめること，(4) 有限制御部の状態を変えること，をまとめて1単位時間で実行する．最初，文字 $x_1, x_2, \ldots, x_n$ を入力テープの左端から n 個のます目に書いて与え (この文字列 $x_1 x_2 \ldots x_n$ が TM への入力である)，

**Turing machine には「チューリング機械」という訳語を充てることが多いが，口述するときにはチューリングマシンということの方が多いし，日本語の「機械」という固いイメージとはかけ離れた概念なので，本書ではあえてチューリングマシンとした．

各ヘッドをテープの左端に置き，ある特定の状態 (初期状態という) から動作を開始する．(1) ～ (4) の基本動作を何回か繰り返した後，指定した状態 (受理状態という) に到達できるか否かで，入力を "認識" する．

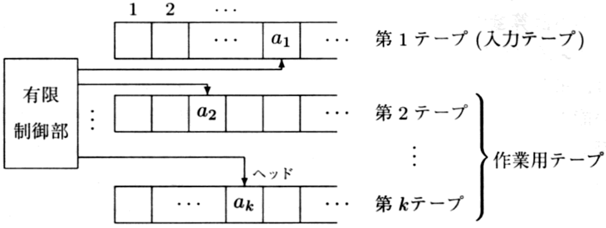


図 **2.1** 多テープ TM

以上述べたことをきちんと定義しよう．***k* テープ TM** (k-tape Turing machine) とは，6 つ組

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

によって定まるシステムである．ここで，各成分は次のようなものである:

(1) Q は空でない有限集合で，Q の元を**状態** (state) という．

(2) Σ と Γ は有限のアルファベットで，$\Sigma \subseteq \Gamma$, $\flat \in \Gamma - \Sigma$ である．Σ を**入力アルファベット** (input alphabet)，Γ を**テープアルファベット** (tape alphabet) という．$\flat$ は**空白記号** (blank symbol) である．

(3) δ は $(Q - F) \times \Gamma^k$ から $Q \times (\Gamma \times \{-1, 0, 1\})^k$ への部分関数である．δ を M の**遷移関数** (transition function) とか，**動作関数** (next move function) という．

(4) q_0 は Q の元で，**初期状態** (initial state) という．

(5) F は Q の部分集合で，F の元を**受理状態** (accepting state) とか，最終状態 (final state) という．

$Q \times (\Gamma^* \times \mathbf{N})^k$ の元を**様相** (configuration) とか，**時点表示** (instantaneous description) といい，**ID** と略記する．ID は TM が現在どのような計算状況にあるかを表わすもので，

$$(q,\ (\alpha_1, h_1), (\alpha_2, h_2), \ldots, (\alpha_k, h_k))$$

は，現在の状態が q, 第 i テープの内容が $\alpha_i \flat\flat\flat \cdots$, 第 i テープのヘッドの位置 (テープの左端から数えて何個目のこまの上にヘッドがあるか) が h_i であることを表わす ($i = 1, 2, \ldots, k$)．このように，テープの内容を表わすときは，右に無限につづく空白は省略する．ただし，必要なら α_i の末尾に空白記号列 $\flat\flat \cdots \flat$ を補い，ヘッド位置 h_i がつねに α_i の上にあるようにする．

TM M の動作を記述するために，ID 間の 2 項関係 $\vdash_M$ を次のように定義する (M が分かっているときには，$\vdash_M$ を $\vdash$ と略記する).

$$I_1 = (q,\ (x_1a_1y_1, h_1), \ldots, (x_ka_ky_k, h_k)),$$
$$\text{ただし，}\ |x_ia_i| = h_i,\ x_i, y_i \in \Gamma^*,\ a_i \in \Gamma\ (1 \le i \le k)$$

を M の ID とする (すなわち，M の現在の状態は q で，第 i テープのヘッドは記号 a_i を読んでいる). このとき，

$$\delta(q, a_1, \ldots, a_k) = (p,\ (b_1, d_1), \ldots, (b_k, d_k))$$

であるならば

$$I_1 \vdash_M I_2$$

であると定義する．ここで，

$$I_2 = (p,\ (x_1b_1y_1, h_1 + d_1), \ldots, (x_kb_ky_k, h_k + d_k)),$$

かつ，すべての i について $h_i + d_i \ge 1$ でなければならない．すなわち，M は状態を p に変え，各 a_i を b_i に書き換え，各ヘッドを d_i だけ動かす．$h_i + d_i \ge 1$ は，M がテープの左端からはみ出ないことを意味する．$\vdash_M$ の n 乗を $\vdash_M^n$ で，$\vdash_M$ の反射推移閉包を $\vdash_M^*$ で表わす．

入力 $x \in \Sigma^*$ に対して，

$$(q_0,\ (x, 1), (\flat, 1), (\flat, 1), \ldots, (\flat, 1))$$

を**初期 ID** (initial ID) といい，状態が受理状態である ID を**受理 ID** (accepting ID) という．I_0 が入力 x に対する初期 ID のとき，ID の列

$$I_0 \vdash_M I_1 \vdash_M \cdots \vdash_M I_m \quad \text{または} \quad I_0 \vdash_M^m I_m$$

を入力 x に対する**計算** (computation) といい，とくに，I_m が受理 ID のとき**受理計算** (accepting computation) という．m をこの計算の**長さ**とか**ステップ数**という．どの I_j $(0 \le j \le m)$ においても ヘッドの位置 $\le s$ であるとき，この計算で**使用する領域**はたかだか s であるという．入力 x の受理計算が存在するとき，M は x を**受理** (accept) すると定義する．したがって，x が M に受理されない (このとき，M は x を**拒否** (reject) するという) のは，受理 ID に到達しないまま計算が終了する ($\vdash_M$ によって移ることのできる ID がそれ以上存在しない) 場合か，受理 ID に到達しない計算が無限につづく場合である．M が受理する語の集合を $L(M)$ で表わす:

$$L(M) = \{x \in \Sigma^* \mid I \vdash_M^* J,\ I \text{ は } x \text{ に対する初期 ID},\ J \text{ は受理 ID}\}.$$

M は言語 $L(M)$ を受理する，ともいう．

$s(n), t(n)$ を $\mathbf{N}$ から $\mathbf{N}$ への関数とする．長さ n の任意の入力に対して，M の受理計算のステップ数が $t(n)$ 以下であるとき (M に受理されない入力につ

いては考慮しない), M は $t(n)$ **時間限定** ($t(n)$ time bounded) であるとか M の**時間量** (time complexity) は $t(n)$ であるという[††]. また, 長さ n の任意の入力に対して, 受理計算の任意の時点で M が使用する領域がたかだか $s(n)$ であるとき, M は $s(n)$ **領域限定** ($s(n)$ space bounded) であるとか M の**領域量** (space complexity) は $s(n)$ であるという. いずれの場合も, 入力が受理されるときには M は無限ループに入らないことに注意する (演習問題 5 参照).

例 2.3 言語 $\mathrm{WcW} = \{wcw \mid w \in \{a,b\}^*\}$ を受理する 2 テープ TM $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を構成しよう.

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\},\ \Sigma = \{a,b,c\},\ \Gamma = \{\#, a, b, c, \flat\},\ F = \{q_4\}$$

とし, 動作関数を次のように定義する (いくつかのグループに分けて説明する). はじめに, 入力が ε でない場合の動作を定義する.

(i) 各 $A \in \Sigma$ に対して $\delta(q_0, A, \flat) = (q_1, (A, 0), (\#, 1))$.

このステップにおいて M がすることは, 第 2 テープの左端のこまに # を書いて状態を q_1 に変えることである. この # は, (iii) において第 2 テープの左端を判定するために使われる.

(ii) $\delta(q_1, a, \flat) = (q_1, (a, 1), (a, 1))$,
$\delta(q_1, b, \flat) = (q_1, (b, 1), (b, 1))$,
$\delta(q_1, c, \flat) = (q_2, (c, 0), (\flat, -1))$,
$\delta(q_1, \flat, \flat) = (q_5, (\flat, 0), (\flat, 0))$.

状態 q_1 において M は, 最初の c が出現するまで第 1 テープの内容を 1 文字ずつ第 2 テープにコピーする (コピーした結果を u とする). もし入力が c を含んでいなければ状態を q_5 に変える (M は状態 q_5 に入るとそれ以上動けなくなるので, これは入力を拒否することを意味する). c を含んでいた場合の入力を ucv ($u \in \{a,b\}^*$) とする.

(iii) $\delta(q_2, c, a) = (q_2, (c, 0), (a, -1))$,
$\delta(q_2, c, b) = (q_2, (c, 0), (b, -1))$,
$\delta(q_2, c, \#) = (q_3, (c, 1), (\#, 1))$.

q_2 は, 第 2 テープの内容を変えずに第 2 ヘッドだけをテープ左端に戻すための状態である (第 1 テープのヘッドは動かさない). 第 2 テープの左端には (i) で # を書いておいたので, 第 2 テープのヘッドが # を読んだら状態を q_3 に変え, 両テープともヘッドを右に 1 こま動かす. 第 1 テープのヘッドは v の左

[††] TM は入力を読み切る必要があるため, 必ず $t(n) \geq n+1$ である. また, $\log n$ のように, $n = 0$ で定義されていなかったり整数でない値をとったりする関数も考えるが, $\log 0 = 0$ と定義し, 時間量 $t(n)$ は $\max\{n+1, \lceil t(n) \rceil\}$ の省略形, 領域量 $s(n)$ は $\max\{1, \lceil s(n) \rceil\}$ の省略形であると考える.

端の記号の上に来る．第2テープには $\#u$ が書いてあり，ヘッドは u の左端の記号の上に来る．

(iv) $\delta(q_3, a, a) = (q_3, (a, 1), (a, 1))$,
$\delta(q_3, b, b) = (q_3, (b, 1), (b, 1))$,
$\delta(q_3, \flat, \flat) = (q_4, (\flat, 0), (\flat, 0))$,
上記以外の $A, B \in \{a, b, c, \flat\}$ に対して $\delta(q_3, A, B) = (q_5, (A, 0), (B, 0))$.

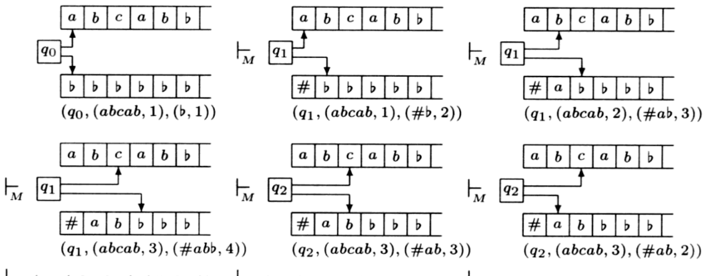
q_3 では，$u, v \in \{a, b\}^*$ かつ $u = v$ であるかどうかを調べる．すなわち，第1テープ上の v の部分と，(iii) において第2テープに書いた u とが等しいかどうかを左から1こまずつ調べる．等しい記号がつづく限り状態 q_3 を保ち，$u = v$ であることがわかったら (そのとき，両ヘッドはともに $\flat$ を読んでいる) 状態 q_4 に入り，入力を受理する．それ以外の場合は状態 q_5 に入り，入力を拒否する．

(i)〜(iv) は，$u = v = \varepsilon$ の場合も正しく機能することに注意する．入力が空語 ε の場合は，(i) の動作をせず，ただちに入力を拒否する:

(v) $\delta(q_0, \flat, \flat) = (q_5, (\flat, 0), (\flat, 0))$.

入力 $abcab$ に対する計算を図 2.2 に示す．

入力 ucv ($u \in \{a, b\}^*, v \in \{a, b, c\}^*$) に対して M の計算はたかだか $2|u| + |v| + 4 \leq 2|ucv| + 2$ ステップ以内に終了するから，M の時間量は $2n + 2$ (M は $2n + 2$ 時間限定 TM) である．また，M のヘッド位置が $|ucv| + 2$ を越えることはないので，M の領域量は $n + 2$ (M は $n + 2$ 領域限定 TM) である．　□



$\vdash_M (q_2, (abcab, 3), (\#ab, 1))$ $\vdash_M (q_3, (abcab, 4), (\#ab, 2))$ $\vdash_M (q_3, (abcab, 5), (\#ab, 3))$
$\vdash_M (q_3, (abcab\flat, 6), (\#ab\flat, 4))$ $\vdash_M (q_4, (abcab\flat, 6), (\#ab\flat, 4))$

図 2.2　入力 abcab に対する計算

上例で見たように，kテープ TM において，「どのテープ記号 $a_1, \ldots, a_k$ に対しても $\delta(q, a_1, \ldots, a_k)$ が定義されていない」ような状態 $q \in Q - F$ のことを**拒否状態** (rejecting state) と呼ぶ．受理状態または拒否状態に入り，それ以上動

けなくなるとき，M は**停止** (halt) するという．

オフライン TM

上述の TM では，入力テープも記憶領域として扱った．TM は入力を必ず読み切るという仮定より (時間量の定義に関する脚注参照)，どんな TM が使用する領域量も $n+1$ 以上である (問 2.3 参照)．しかしこれでは，入力とは別に計算のために本質的に使う領域量が入力の長さより小さい計算をそれ以上分類することができない．

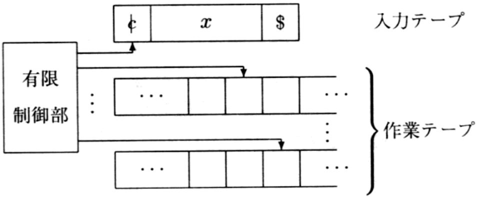


図 2.3 オフライン TM

そこで，領域量が n より小さい場合を議論するときには，オフライン型と呼ばれる TM を用いる．**オフライン TM** (off-line TM) では入力テープは読み出し専用に使われ，書き込みはできない．Σ を入力アルファベットとするとき，入力 $x \in \Sigma^*$ は前後に終止符 (endmarker) ¢ と \$ を付けた文字列 ¢x\$ として与えられる (図 2.3 参照)．ただし，¢, \$ $\notin \Sigma$ とする．この終止符によって，入力テープのヘッドが入力の端に来たかどうかを検出できる．

オフライン TM M においては，長さ n の任意の入力に対して，受理に至る計算の過程においてどの作業テープのヘッド位置も $s(n)$ を越えないとき，領域量は $s(n)$ (M は $s(n)$ 領域限定) であるという．

問 2.3「入力を読み切らなければならない」という仮定をおかない場合，n 未満の領域量や時間量しか使わない TM が受理する言語はどのようなものになるか？

例 2.4 アルファベット $\{a, b\}$ 上の言語 AnBn=$\{a^n b^n \mid n \geq 0\}$ を受理する 1 作業テープ・オフライン TM M を考えよう (M は入力テープ以外に作業テープ 1 本を持つ)．1.6 節で述べた擬似プログラムによって M の動作を記述する．

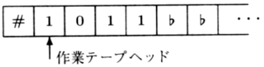


図 2.4 M の作業テープ

はじめに，作業テープに 2 進数 $n \in \{0,1\}^*$ がすでに書いてあり (左側が下位の桁で，テープの左端のこまには記号 # が書いてある)，ヘッドが n の左端のこまの上にあるとき (図 2.4: この例では n の値は 13)，これに 1 を足すことを行なう操作 (M の動作の一部) を手続き add1 として記述する．

```
procedure add1()
begin
    while 作業テープヘッドが読んでいる記号 = '1' do
        '1' を '0' に書き換え，作業テープヘッドを右に 1 こま動かす
    end;
    /* この時点で，作業ヘッドは '♭' または '0' が書かれたこまの上にある */
    作業ヘッドの位置に '1' を書く;
    作業ヘッドを元の位置 (#の右隣りのこま) に戻す
        /*
            この部分を丁寧に書くと以下のようになる:
                while 作業ヘッドが読んでいる記号 ≠ '#' do
                    作業ヘッドを左に 1 こま動かす
                end;
                作業ヘッドを右に 1 こま戻す
        */
end
```

同様に，作業テープ上に書かれた 2 進数から 1 を引く手続き subtract1 が定義できる (問 2.4)．これらを使うと，M の動作は次のように記述できる．

```
procedure M(x)
/* 入力 x が a^n b^n の形かどうかを判定する TM */
begin
    作業テープの左端のこまに#を書き，両テープのヘッドを右に 1 こま動かす;
    while 入力記号 (入力テープヘッドが読んでいる記号) = 'a' do
        入力ヘッドを右に 1 こま動かす; add1();
    end;
    while 入力記号 = 'b' do
        入力ヘッドを右に 1 こま動かす; subtract1();
    end;
    if 入力記号 = '$' かつ作業テープ上の 2 進数の値 = 0
        then 入力 x を受理 else x を拒否
    end
end
```

例 2.3 で TM を定義した際は動作関数 δ をきちんと示したが，今後はこの例のように，動作関数の詳細を記すことはしない．その方が簡潔で分かりやすいし，この例のような記述から動作関数 δ をきちんと示すことは容易にできるからである．

入力が $x = a^\ell w$ ($\ell \geq 0,\ w \in b\{a,b\}^* \cup \{\varepsilon\}$) のとき，$M$ は作業テープの領域をたかだか (ℓの 2 進数表現の桁数) $+1$ しか使わない．よって，M の領域量は

$\lceil \log_2(n+1) \rceil$ である ($n = |x|$ の関数としてみたとき，最も領域をたくさん使うのは $x = a^n$ のときである)．一方，計算時間が最もかかるのは $x = a^\ell b^m$ ($\ell \geq m$) のときで，M の時間量は $8n$ である (問 2.4)． □

問 2.4 例 2.4 に関する次の問に答えよ．

(1) 手続き subtract1 を定義せよ (引き算した結果，最上位の桁が 0 になったら，記号 '0' ではなく '♭' を書くこと)．

(2) 入力 $x = a^n$ に対する M の時間量は $8n$ であることを示せ．ヒント：ヘッド 2 は，2 進数 n の左から i ビット目 (2^{i-1} の位) を保持するます目を延べで，たかだか $4 \times n/2^{i-1}$ 回しか通過しない．

(3) AnBn を受理する，時間量も領域量も $n+1$ の 2 テープ TM が存在することを示せ．

両方向無限テープ

ここでは，各テープが左右両方向に無限である TM (TM with two-way infinite tapes) について考察する．$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ をそのような k テープ TM としよう (図 2.5 左)．

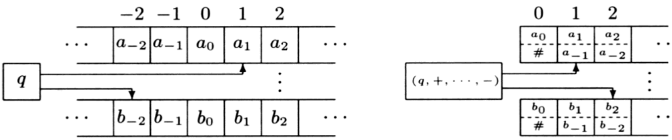


図 2.5 2 方向無限テープ TM (左) と対応する片方向無限テープ TM (右)

各テープのます目には，ある所を原点 0 として番号 $\ldots, -2, -1, 0, 1, 2, \ldots$ が振られている．入力 $x_0 x_1 \ldots x_{n-1}$ ($x_i \in \Sigma$) は，第 1 テープのます目 $0, 1, \ldots, n-1$ に与えられる．M の ID は $Q \times (\Gamma^* \times \mathbf{Z})^k$ の元

$$(q,\ (\alpha_1, h_1), (\alpha_2, h_2), \ldots, (\alpha_k, h_k))$$

である (M は原点より左のます目にもヘッドを移動させ，そこのます目の内容を書き換えることができる)．$\vdash_M$ の形式的な定義は省略する．

両方向無限テープ TM の時間量の定義は片方向無限テープ TM の場合と同じであるが，領域量の定義は次のように変更する: 長さ n の，M に受理されるどんな入力に対しても各テープのヘッド位置が $-\ell$ ($\ell \geq 0$) と r ($r \geq 0$) の間しか動かないとき，M の領域量は $\ell + r + 1$ であると定義する．

定理 2.1 M を両方向無限テープ TM とする．$L(M') = L(M)$ かつ M' の時間量・領域量と M のそれとが等しいような片方向無限テープ TM M' が存在する．

証明 荒っぽく言うと，M' は，M のテープを原点で折り返したようなテープ (図 2.5 右: 各テープを 2 本の**トラック** (track) に分け，正の方向を上のトラックで，負の方向を下のトラックで表わす) を使って M の動作を模倣する．すなわち，M の第 i こま ($i = 0, \pm 1, \pm 2, \ldots$) にテープ記号 a_i が書いてあるとき，M' の第 i こま ($i = 1, 2, \ldots$) にはテープ記号 $(a_i, a_{-i}) \in \Gamma \times \Gamma$ を書く．ただし，第 0 こまには $(a_0, \#)$ を書く．M のテープヘッドが原点より左にあるか右にあるかに従って M' はヘッド位置のこまの上下どちらのトラックが対応しているかを $+, -$ で表わし，M の現在の状態と一緒に有限制御部に記憶しておく．

読者にトラックという概念を理解してもらうために，M' の形式的な定義をしておこう．記述が簡単になるように，$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ が 1 テープ TM である場合を考える (多テープやオフライン型の場合も同様である)．

$$Q' = Q \times \{+, -\} \cup \{q_0'\},\ \Gamma' = \Gamma \cup \Gamma \times (\Gamma \cup \{\#\}),\ F' = F \times \{+, -\}$$

とし，$M' = (Q', \Sigma, \Gamma', \delta', q_0', F')$ とする．δ' を以下のように定義する．

(1) 模倣を始めるに先立って，M' は原点のます目の記号を 2 トラック化し，M と同じ初期状態に設定する: $a \in \Sigma \cup \{\flat\}$ のとき，

$$\delta'(q_0', a) = ((q_0, +), (a, \#), 0).$$

これ以後も，M' は初めて Γ の元に出会うと，それを 2 トラック化する．

(2) M のヘッド位置が原点より右にある場合，M' は上のトラックを見て M の動作を模倣する: $a, b \in \Gamma, d \in \{-1, 0, 1\}$, $\delta(q, a) = (q', a', d)$ のとき，

$$\delta'((q, +), (a, b)) = ((q', +), (a', b), d),$$
$$\delta'((q, +), a) = ((q', +), (a', \flat), d).$$

(3) M のヘッド位置が原点より左にある場合，M' は下のトラックを見て M の動作を模倣する: $a, b \in \Gamma, d_1, d_2 \in \{-1, 0, 1\}$, $\delta(q, b) = (q', b', d_1)$, $\delta(q, a) = (q'', b'', d_2)$ のとき，

$$\delta'((q, -), (a, b)) = ((q', -), (a, b'), -d_1),$$
$$\delta'((q, -), a) = ((q'', -), (a, b''), -d_2).$$

(4) M のヘッド位置が原点にあり，そこから左に動く場合，M' は (上のトラックから) 下のトラックへ切り替えなければならない: $a \in \Gamma$, $\delta(q, a) = (q', a', -1)$ のとき，

$$\delta'((q, +), (a, \#)) = \delta'((q, -), (a, \#)) = ((q', -), (a', \#), 1).$$

(5) M のヘッド位置が原点にあり，そこから右に動く場合，M' は (下のトラックから) 上のトラックへ切り替えなければならない: $a \in \Gamma, d \in \{0, 1\}$, $\delta(q, a) = (q', a', d)$ のとき，

$$\delta'((q,-),(a,\#)) = \delta'((q,+),(a,\#)) = ((q',+),(a',\#),d).$$

明らかに，M' は定理の条件を満たしている． □

片方向無限テープ TM は両方向無限テープ TM の特別の場合 (原点のこまに特殊な記号を書き込んでおき，それより左にヘッドが移動しないようにしたもの) と考えることができるので，言語受理能力や計算量に関して両者をまったく区別する必要がないことをこの定理は示している．

2.3 問題の符号化 (なぜ TM がアルゴリズムなのか)

前節で、TM とは語を入力としてそれを受理するかどうかを判定するだけの機械として定義した．すなわち，TM M はあるアルファベット上の言語 $L(M)$ を認識することしかできない．そのような単純な機械である TM をなぜアルゴリズムと考えることができるのであろうか？ その答は，「扱いたい対象を文字列で表現して TM への入力とする」ところにある．

Ω を対象の集合とし，Σ を有限アルファベットとする．Ω から Σ^* への単射 ($x \neq y \Longrightarrow \sigma(x) \neq \sigma(y)$ を満たす写像) σ のことを**符号化** (encoding) といい，$\sigma(x)$ を $x \in \Omega$ の**コード** (**符号語**, code) と呼ぶ．逆に，$\alpha \in \Sigma^*$ から $\sigma^{-1}(\alpha) \in \Omega$ を求めることを**復号** (decoding) という．以後，σ を明示的に示さず，x のコードを $\overline{x}$ で表わすことが多い．

答として yes/no だけを求める問題を**決定問題** (decision problem) という．例えば，ハミルトン閉路問題「グラフ G が与えられたとき，G はハミルトン閉路を持つか？」，ヒルベルトの第 10 問題「整数係数の多変数多項式 $p(x_1,\ldots,x_n)$ が与えられたとき，$p(x_1,\ldots,x_n)=0$ は自明でない整数解を持つか？」，TM の停止問題「TM M と M への入力 w が与えられたとき，w に対する M の計算は停止するか？」などはいずれも決定問題である．いま，ひとつの決定問題 P を考えよう．P で扱う対象 x (x は P の具体例の 1 つであり，**インスタンス** (instance) ということもある) のうち，答が yes であるものだけを考え，それらのコードを元とする集合を L_P とする:

$$L_P = \{\overline{x} \mid x \text{ は問題 P の対象で}, x \text{ に対する答は yes である}\}.$$

L_P は Σ 上の言語である．言語 L_P を受理する TM は，問題 P を解いているアルゴリズムであると考えることができる: M が $\overline{x}$ を受理 $\Longleftrightarrow$ x に対する答が yes. このように，決定問題は TM による言語の**認識問題** (recognition problem) として定式化することができる．

例 2.5 (1) **自然数の符号化** 自然数は，基数を何にするかによっていろいろな符号化ができる．例えば，10 進数として表現する場合には $\{0,1,\ldots,9\}^*$ の元がコードとなるし，2 進数として表現する場合には $\{0,1\}^*$ の元がコードとなる．また，自然数 n を 1^n で表わす方法を **tally 表現**という．どの表現を用いても大差ないが，本書では 2 進数表現を用いることにし，n のコードを $\overline{n}$ で表わす．ただし，0 は ε で，0 以外は $1\{0,1\}^*$ の元で表わすものとする: $\overline{0}=\varepsilon$, $\overline{1}=1$, $\overline{2}=10$, ..., $\overline{23}=10111$, etc.

(2) **ハミルトン閉路問題** $G=(V,E)$ を有向 (あるいは無向) グラフとし，$V=\{v_1,\ldots,v_n\}$, $E=\{e_1,\ldots,e_m\}$ とする．$\Sigma=\{0,1,\#\}$ をアルファベットとし，頂点 v_i のコード $\overline{v_i}$ を $\overline{v_i}=\#\overline{i}$ で定め，辺 $e_i=(v_j,v_k)$ のコードを $\overline{e_i}=\overline{v_j}\,\overline{v_k}$ で定める．最後に，G のコードは $\overline{G}=\overline{n}\#\#\overline{e_1}\cdots\overline{e_m}$ で定める．ハミルトン閉路問題は，言語

$$L_{Hamilton}=\{\overline{G}\mid G\text{ はハミルトン閉路を持つグラフ}\}$$

の認識問題として定式化される．

(3) **ヒルベルトの第 10 問題** 変数 x_i を $\overline{x_i}=\#\overline{i}$ で表わし，例えば，整数係数の不定方程式 $p: 3x_1^5x_3+x_2^2x_3-8x_1^3+7=0$ をアルファベット $\{0,1,\#,+,-,\uparrow\}$ 上の語 $\overline{p}=\overline{3}\,\overline{x_1}\uparrow\overline{5}\,\overline{x_3}\uparrow\overline{1}+\overline{1}\,\overline{x_2}\uparrow\overline{2}\,\overline{x_3}\uparrow\overline{1}-\overline{8}\,\overline{x_1}\uparrow\overline{3}+\overline{7}$ で表わすと，ヒルベルトの第 10 問題とは，言語

$$L_{Hilbert}=\{\overline{p}\mid p\text{ は自明でない整数解を持つ整数係数不定方程式}\}$$

の認識問題のことである．

(4) **TM の停止問題** (halting problem for TMs) 後で使う必要上，ここでは TM を $\{0,1\}$ 上の語として符合化する．$M=(Q,\{0,1\},\Gamma,\delta,q_1,F)$ を 1 テープ TM とし，$w\in\{0,1\}^*$ を M への入力とする．$Q=\{q_1,\ldots,q_r\}$, $\Gamma=\{a_1,\ldots,a_s\}$, $F=\{q_2\}$, $\Delta_M=\{\alpha_1,\ldots,\alpha_t\}$ とする．ただし，$\flat=a_1$, $0=a_2$, $1=a_3$ とする．M は集合

$$\Delta_M=\{(q_i,a_j,q_k,a_\ell,d)\in Q\times\Gamma\times Q\times\Gamma\times\{-1,0,1\}\mid\delta(q_i,a_j)=(q_k,a_\ell,d)\}$$

と同一視することができる．Δ_M の各元のコードを

$$\overline{(q_i,a_j,q_k,a_\ell,d)}=1^i01^j01^k01^\ell01^{d+2}$$

と定義し，M のコードを $\overline{M}=000\overline{\alpha_1}00\overline{\alpha_2}00\cdots00\overline{\alpha_t}000$ として定義する．言語

$$L_{halting}=\{\overline{M}w\mid\text{入力 } w \text{ に対する } M \text{ の計算は停止する}\}$$

の認識問題のことを TM の停止問題という．ここでは $w\in\{0,1\}^*$ としているが，w が一般のアルファベット Σ 上の語の場合も，対 (M,w) を $\{0,1\}$ 上の語として容易に符号化できる．また，上記の TM では $F=\{q_2\}$ としているが，このような仮定をしても一般性は失われない (問 2.6). □

問題を解くのにかかる計算量は問題の大きさ (サイズ) に依存すると考えるのが自然である．したがって，符号化はもともとの問題のサイズをあまり大きく変えないようなものであることが望まれる．そのような意味で，もとの問題 x のサイズが n であるとき，そのコード $\overline{x}$ が

$$|\overline{x}| = O(n \log n)$$

を満たす符号化を**簡潔な符号化**という．グラフのサイズとしては頂点数 (あるいは，辺数) を，不定方程式のサイズとしては項数を，TM M のサイズとしては $|\Delta_M|$ を考えるのが適当であろう．上例のどの符号化も簡潔な符号化である．

問 2.5 ここでは符号化用アルファベットを $\Sigma = \{0,1\}$ に固定する符号化を考える．まず，準同型写像とは何かを定義する．Γ, Δ をアルファベットとするとき，写像 $h : \Gamma^* \to \Delta^*$ が

(1) $h(\varepsilon) = \varepsilon$,

(2) 任意の $x_1 x_2 \cdots x_n$ (各 $x_i \in \Gamma$) に対して

$$h(x_1 x_2 \cdots x_n) = h(x_1) h(x_2) \cdots h(x_n)$$

を満たすとき，h を Γ から Δ への**準同型写像** (homomorphism) という．

Σ をアルファベット，$\#$ を Σ の元でない記号とし，m を正整数とする．h は $\Sigma \cup \{\#\}$ から Σ への単射な準同型写像で，任意の $a \in \Sigma$ に対して $|h(a)| = m$ であるとする．$\langle x, y \rangle = h(x\#y)$ によって定まる 2 変数関数 $\langle\ \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ を**対関数** (pairing function) という．$\langle\ \rangle$ を次のように多変数に拡張する: $n \geq 3$ のとき，

$$\langle x_1, x_2, \ldots, x_n \rangle = \langle\langle\langle \ldots \langle x_1, x_2 \rangle, \ldots \rangle, x_n \rangle, n \rangle.$$

符号の中に n も組み込むのは，復号が一意的に定まるようにするためである．例えば，$\Sigma = \{0,1\}$, $h(0) = 00$, $h(1) = 11$, $h(\#) = 01$ とすると，$\langle 00, 11 \rangle = h(00\#11) = h(0)h(0)h(\#)h(1)h(1) = 0000011111$ である．

(1) $\langle 0, 1, 10, 11 \rangle$ を求めよ．

(2) $\langle\ \rangle$ は簡潔な符号ではないことを示せ．

問 2.6 任意の TM M に対して，$L(M) = L(M')$ となる TM M' で受理状態が 1 個だけ，かつ，テープ記号が $0, 1, \flat$ だけのものが存在することを示せ．

2.4 Turing 計算可能関数

2.1 節で，計算可能性に関するさまざまな計算モデルを考察し，もう一つの計算モデルとして 2.2 節で TM を導入した．前節までは TM を言語の認識装置として扱ったが，この節では TM を関数を定義する装置と見て，TM によって定義される関数が 2.1 節で考察した計算可能関数と一致することを示す．

この節では，自然数 n に対して $\overline{n}$ は，n の tally 表現を表わすものとする．すなわち，$\overline{0} = \varepsilon$, $\overline{n} = \overbrace{11 \cdots 1}^{n}$ とする．

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を多テープ TM とする．M のテープのうちの 1 本 (入力テープ以外のどのテープでもよい) を**出力テープ** (output tape) と定める．

出力テープは書き込み専用 (そこに書き出された内容が TM の以後の動作に影響することがない) とし，出力テープのヘッドが左に動くことはないものとする．M に入力 x が与えられたとき，x に対する計算が停止し，停止したときの出力テープの内容 (計算開始時のヘッド位置と停止時のヘッド位置の間に書き出された出力テープ上の文字列のこと．空白記号は含まないものとする) が y のとき，y を入力 x に対する**出力** (output) という．入力 x に対する M の出力を $M(x)$ で表わす．x に対する計算が停止しないなら，$M(x)$ は定義されない．M は Σ^* から Γ^* への部分関数と考えることができる．このような TM のことを**変換機** (transducer) という．TM を変換機と考える場合にはオフライン型であるとし，また，出力テープ上で使われる領域は領域量にカウントしない．

例 2.6 次の手続きに示したオフライン TM M は，入力 $\overline{x}0\overline{y}$ $(x, y \in \mathbf{N})$ に対して $\overline{x+y}$ を出力する:

```
procedure M(x̄0ȳ)
begin
    while 入力記号 = '0' or '1' do
        if 入力記号 = '1' then '1' を出力 end;
        入力ヘッドを右に 1 こま動かす
    end
end
```

M は自然数の加法 $(x, y) \mapsto x + y$ の計算を行なっていると見ることができる．M の領域量は 1 である (p.42 の脚注参照)． □

$n \in \mathbf{N}$ とし，φ を $\mathbf{N}^n$ から $\mathbf{N}$ への (部分) 関数とする．

$$\begin{cases} M(\overline{x_1}0\overline{x_2}0\cdots0\overline{x_n}) = \overline{y} \iff \varphi(x_1, x_2, \ldots, x_n) = y \\ \text{入力 } \overline{x_1}0\overline{x_2}0\cdots0\overline{x_n} \text{ に対し } M \text{ は停止しない} \iff \varphi(x_1, x_2, \ldots, x_n) \text{ は未定義} \end{cases}$$

が任意の $x_1, x_2, \ldots, x_n$, $y \in \mathbf{N}$ に対して成り立つような TM M が存在するならば，φ は **Turing 計算可能** (Turing computable) であるという．上例で見たように，加法 $(x, y) \mapsto x + y$ は Turing 計算可能関数である．

問 2.7 乗法 $(x, y) \mapsto xy$ が Turing 計算可能関数であることを示せ．とくに，$(x, y) \mapsto xy$ を計算する TM で $\log_2 n$ 領域限定であるものを示せ．また，$(x, y) \mapsto xy$ を計算する $O(n)$ 時間限定 TM は存在しないことを示せ．

命題 2.2 自然数の上の (部分) 関数 f が Turing 計算可能であることと f が帰納的 (部分) 関数であることとは同値である (演習問題 6 参照)．

2.5 TM と句構造文法

TM が受理する言語を**帰納的可算言語** (recursively enumerable language) という．「帰納的」という言葉は「アルゴリズムが存在する」の同義語としてしばしば用いられる．「可算」とは「数え上げる (枚挙する) ことができる」という意味である．TM が受理する言語をなぜこのように呼ぶのかは以下の事実に由来する．

M を出力テープを持つ TM とする．M のテープアルファベットを $\Gamma \cup \{\#\}$ とする．M に空語 ε を入力として与えると出力テープ上に語

$$w_1\#w_2\#w_3\#\cdots \quad (\text{各 } w_i \in \Gamma^*)$$

を生成しつづける ($w_1\#w_2\#\cdots\#w_k$ を生成して停止する場合も含む) とき，M は言語 $G(M) = \{w_1, w_2, w_3, \ldots\}$ を**生成** (generate) するという．M が停止する場合，$G(M)$ は有限集合である．M は言語 $G(M)$ の元を枚挙していると見ることができる．

定理 2.2 言語 L が帰納的可算集合であることと L が TM によって生成されることとは同値である．

証明 ($\Longrightarrow$) $L \subseteq \Sigma^*$ を受理する TM を M とする．L を生成する TM M' を構成したい．Σ^* の語を適当な系統的方法で列挙し，列挙された順に M への入力として M の動作を模倣し，M が受理したら M' はそれを出力テープに書き出し，M が拒否したら次の順番の Σ^* の語について M の模倣を行なう，という方法では，途中で無限ループに陥る語があると生成がそこでストップしてしまう．

そこで，M' を次のように作る．Σ^* の語を例えば標準順序 (1.3 節参照) $w_1, w_2, \ldots, w_i, \ldots$ で並べる．w_i を入力としたときの M の動作を初期 ID から j ステップだけ行なうことを (i,j) 模倣と呼ぶことにする．M' は $(1,1)$, $(1,2)$, $(2,2)$, $(1,3)$, $(2,3)$, $(3,3)$, $(1,4), \ldots$ の順に (i,j) のコードを作業用テープ上に生成し，この順に (i,j) 模倣を行なう．これによって，M' は途中で無限ループに陥ることなく M が受理するすべての入力を生成することができる．

($\Longleftarrow$) L を生成する TM を M とする．L を受理する TM M' を次のように構成する．入力 w が与えられたとき，M' は入力とは無関係に M の動作を模倣する．M が出力テープに $w_i\#$ を書き出すたびに M' は模倣を一旦中止して w と w_i が等しいかどうかを調べ，等しかったら入力を受理して停止し，等しくなかったら M の模倣を再開する． □

問 2.8 以後，定理 2.2, のように，TM の動作の記述をかなり簡略に述べる．定理 2.2 で定義されている 2 つの TM M' を動作関数によってきちんと定義せよ．

一般に，TM はすべての入力に対してその計算が停止するわけではない．すなわち，TM M が受理しない語 $w \notin L(M)$ には，M が w に対する計算を開始してから有限回のステップの後停止して拒否するものと，計算が無限ループに陥ってしまうために受理されないものとの 2 種類がある．後者のタイプの拒否の仕方は排除できるであろうか？ 答は no である．どんな入力に対しても必ず停止する TM を**停止性 TM** (halting TM) といい，停止性 TM によって受理される言語を**帰納的言語** (recursive language) という．帰納的ではない帰納的可算言語が存在することを 4 章で示す．

本章ではさまざまな計算モデルを取り上げているが，最後に句構造文法について述べる．句構造文法は，米国の言語学者チョムスキーによって 1956 年頃提唱された，**生成文法** (generative grammar) と総称される「言語を生成する機構である文法の数学的モデル」の一つである[§]．それまで統計的・定量的な研究が主だった自然言語 (日本語や英語などのように日常的に人間が使う言語のこと) の研究に初めて形式的・数学的手法を取り入れ，**形式言語理論** (formal language theory) と呼ばれる，言語の定性的な研究の新しい流れを作った．その後，形式言語理論は言語を受理する機構である**オートマトン** (automaton: TM のサブクラスやその変種のこと) と表裏一体で研究され，コンピュータの発達に伴い折しも盛んになったプログラミング言語やコンパイラ等の研究においても強力なツールの一つとなった．計算量理論は形式言語・オートマトン理論と密接な関連を持ち，互いに影響を及ぼしあって発展してきた．本書でも，後の章で言語理論から多くの例を取り上げる．

句構造文法 (phrase-structure grammar) とは 4 つ組

$$G = (N, \Sigma, P, S)$$

のことである．ここで，

(1) N は有限アルファベットで，N の元を**非終端記号** (nonterminal symbol) という．

(2) Σ は N と互いに素な有限アルファベットで，Σ の元を**終端記号** (terminal symbol) という．$V = N \cup \Sigma$ とおく．

(3) P は $V^+ \times V^*$ の有限部分集合で，P の元を**プロダクション** (production) とか，**書き換え規則** (rewriting rule) という．プロダクション (x, y) は通常 $x \to y$ と書く．

(4) S は N の特別な元で，**開始記号** (start symbol) という．

V^* 上の 2 項関係 $\underset{G}{\Rightarrow}$ (G が明らかな場合は単に $\Rightarrow$ と書く) を

[§] N.Chomsky, Three models for the description of languages, *IRE Trans. Inform. Theory* 2, pp.113-124, 1956.

$$uxv \underset{G}{\Rightarrow} uyv \quad \overset{\text{def}}{\Longleftrightarrow} \quad x \to y \in P \qquad (u, v \in V^*)$$

で定義する．$w_0 \underset{G}{\Rightarrow} w_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} w_n$ であるとき，この過程を**導出** (derivation) と呼び，w_n は w_0 から導出 (derive) されるという．開始記号 S から導出される Σ^* の語の集合を $L(G)$ で表わし，$L(G)$ は G によって**生成** (generate) されるという ($\overset{*}{\underset{G}{\Rightarrow}}$ は $\underset{G}{\Rightarrow}$ の反射推移閉包):

$$L(G) = \{w \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}.$$

句構造文法によって生成される言語を**句構造言語** (phrase-structure language) という．句構造文法は **0 型文法** (type 0 grammar) とか，**semi-Thue システム**とも呼ばれる (後述の文脈依存文法，文脈自由文法，正則文法をそれぞれ 1 型，2 型，3 型の文法とも呼ぶ)．

例 2.7 (1) $G_1 = (\{S\}, \{a, b\}, \{S \to aS,\ S \to bS,\ S \to \varepsilon\}, S)$ とする．$L(G_1) = \{a, b\}^*$ である．実際，例えば aab は次のように S から導出される:

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aabS \Rightarrow aab.$$

G_1 のように，すべてのプロダクションが $A \to xB$ または $A \to x$ ($A, B \in N$, $x \in \Sigma \cup \{\varepsilon\}$) という形をしている文法を**正則文法** (regular grammar) とか**右線形文法** (right-linear grammar) といい，正則文法が生成する言語を**正則集合** (regular set) とか**正規言語**という．したがって，$\{a, b\}^*$ は正則集合である．

(2) $G_2 = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to \varepsilon\}, S)$ とする．G_1 は言語 AnBn $= \{a^n b^n \mid n \geq 0\}$ を生成する．G_2 のように，すべてのプロダクションが $A \to x$ ($A \in N$, $x \in V^*$) という形をしている文法を**文脈自由文法** (context-free grammar) といい，文脈自由文法が生成する言語を**文脈自由言語** (context-free language) という．AnBn は文脈自由言語である (AnBn は正則集合でないことが知られている)．

(3) $G_3 = (\{S, X, Y, Z, A, B\}, \{a, b, c\}, P, S)$ を考える．ただし，P は次のプロダクションからなるものとする:

(i)	$S \to XY$	(ii)	$X \to aXa$	(iii)	$X \to bXb$	(iv)	$X \to cZ$
(v)	$aY \to AY$	(vi)	$bY \to BY$	(vii)	$aA \to Aa$	(viii)	$bA \to Ab$
(ix)	$aB \to Ba$	(x)	$bB \to Bb$	(xi)	$ZA \to aZ$	(xii)	$ZB \to bZ$
(xiii)	$ZY \to \varepsilon$						

説明の便宜上，プロダクションに番号を付けた．例えば，$aabcaab$ は次のように生成される:

$$\underline{S} \Rightarrow \underline{X}Y \Rightarrow a\underline{X}aY \Rightarrow aa\underline{X}aaY \Rightarrow aab\underline{X}baaY \Rightarrow aabcZba\underline{aY}$$

$$\Rightarrow aabcZb\underline{aA}Y \Rightarrow aabcZ\underline{bA}aY \Rightarrow aabc\underline{ZA}baY$$
$$\Rightarrow aabcaZb\underline{aY} \stackrel{*}{\Rightarrow} aabcaa\underline{ZB}Y \Rightarrow aabcaab\underline{ZY} \Rightarrow aabcaab.$$

G_3 はまず (i) により XY を導出し，つづいて (ii)〜(iv) により

$$a_1 \cdots a_n cZa_n \cdots a_1 Y \quad (n \geq 0,\ 各\ a_i \in \{a, b\})$$

を導出する．(v) または (vi) により Y の左隣りの a_1 を A ($a_1 = a$ のとき) または B ($a_1 = b$ のとき) に書き換え，(vii)〜(x) によりそれを Z の右隣りまで移動し，(xi) または (xii) により a_1 を Z の左隣りに作る．以上で Y の左隣りにあった a_1 を Z の左隣りに移動する操作を終える．この操作を繰り返すと $a_1 \cdots a_n a_1 \cdots a_n ZY$ が導出され，最後に (xiii) により ZY が消去される．G_3 が $\{a, b, c\}^*$ の語を導出するのはこのような場合しかない．よって，$L(G_3) = \mathrm{WcW} = \{wcw \mid w \in \{a, b\}^*\}$ である．

すべてのプロダクション $x \to y$ が $|x| \leq |y|$ という条件を満たす文法を**文脈依存文法** (context-sensitive grammar) といい，文脈依存文法が生成する言語を**文脈依存言語** (context-sensitive language) という．G_3 は (xiii) のプロダクションを除くと文脈依存文法の条件を満たしている．実は WcW は文脈依存言語である．また，WcW は文脈自由言語でないことが知られている． □

問 2.9 (1) $\mathrm{WW}^R = \{ww^R \mid w \in \{a, b\}^*\}$ は文脈自由言語であることを示せ．
(2) 例 2.7(3) の文法 G_3 を，WcW を生成する文脈依存文法となるように修正せよ．また，$\mathrm{WW} = \{ww \mid w \in \{a, b\}^*\}$ を生成する文脈依存文法も示せ．

3 章で非決定性という概念を導入し，TM と非決定性 TM とが等価であることを証明する．次の命題は非決定性 TM を用いると容易に示すことができる (第 3 章演習問題 1)．

命題 2.3 言語 L が TM で生成されることと L が句構造文法で生成されることとは同値である．

演 習 問 題 2

1 while 文を含まない while プログラムを **loop プログラム**といい，loop プログラムによって定義できる関数を **loop 関数**という．loop 関数と原始帰納的関数とは一致することが知られている．

while 文を含まない文 S に対して，$\mathrm{nest}(S)$ を次のように再帰的に定義する:

(1)　S が代入文ならば $\mathrm{nest}(S) = 0$.
(2)　S が $S_1; S_2$ ならば $\mathrm{nest}(S) = \max\{\mathrm{nest}(S_1), \mathrm{nest}(S_2)\}$.
(3)　S が **loop** x **do** S_1 **end** ならば $\mathrm{nest}(S) = \mathrm{nest}(S_1) + 1$.

nest(S) は，S における **loop** $\cdots$ **end** の入れ子構造の深さを表わす．loop プログラム $f(x_1,\ldots,x_n) \to y$ **begin** S **end** の**最大ネスト**とは nest(S) のことをいう．$n \in \mathbf{N}$ に対して，loop 関数のクラス $\mathcal{L}_n$ を次のように定義する:

$$\mathcal{L}_n \;=\; \{f \mid f \text{ を定義する loop プログラムで，最大ネストが } n \text{ 以下のものが存在する}\}.$$

(1) 次のことを示せ:

(a) $\mathrm{add} \in \mathcal{L}_1 - \mathcal{L}_0$ (b) $\mathrm{mult}, \mathrm{div}, \mathrm{mod} \in \mathcal{L}_2 - \mathcal{L}_1$

ただし，$\mathrm{mod}(x,y) =$「x を y で割った余り ($y = 0$ のときは 0)」である．

(2) $f_0(0) = 1$, $f_0(1) = 2$, $x > 1$ のとき $f_0(x) = x + 2$ とし，$i \geq 0$ に対して $f_{i+1}(x) = \underbrace{f_i(f_i(\cdots f_i}_{x}(x)\cdots))$ と定義する．$f_i \in \mathcal{L}_i$ であることを示せ (実は，$f_i \in \mathcal{L}_i - \mathcal{L}_{i-1}$ である)．

2 次のように定義される関数 $A : \mathbf{N}^2 \to \mathbf{N}$ を**アッケルマン関数** (W.Ackermann, 1928 年) という:

$$\begin{cases} A(0,y) = y+1 \\ A(x+1,0) = A(x,1) \\ A(x+1,y+1) = A(x, A(x+1,y)). \end{cases}$$

(1) アッケルマン関数を定義する while プログラムを書け．

(2)

$$A(4,y) > 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} 2 \text{ が } y \text{ 個}$$

であることを示せ．アッケルマン関数は原始帰納的ではない帰納的関数の一例である．実際，$A(x,y)$ は非常に増加速度が速い関数で，どんな原始帰納的関数 $g(x)$ に対しても，十分大きな x をとれば $A(x,x) > g(x)$ とすることができる．

3 **λ定義可能関数** (λ-definable function) は A.Church により導入された[¶]．**変数**を $x, x_1, x_2, \cdots$ 等の記号で表わす．**λ式** (λ-term) およびその中に現れる変数が**束縛** (bound) されているか**自由** (free) であるかを，次のように再帰的に定義する (同じ記号で表わされている変数でも，現れている位置によって束縛変数か自由変数かが異なりうる)．

(1) x が変数ならば x はλ式であり，変数 x はλ式 x の中で自由である．

(2) M がλ式で x が変数ならば $(\lambda x.M)$ はλ式であり，M の中に現れる x のうち自由なものはこの λx の x によって $(\lambda x.M)$ において束縛されている．λx の x 自身も束縛変数である．それ以外の変数が $(\lambda x.M)$ において束縛されているか自由であるかは，それが M において束縛されているか自由であるかと同じである．$(\lambda x.M)$ を x に関する M の**関数抽象** (functional abstraction) という．

(3) M, N がそれぞれλ式ならば (MN) はλ式であり，(MN) に現れる各変数が束縛されているか自由であるかは，それが M あるいは N において束縛されているか自由であるかと同じである．(MN) を N に対する M の**関数適用** (functional application) という．

荒っぽく言うと，$(\lambda x.M)$ は x に M を対応させる関数 $x \mapsto M$ を表わし，(MN) は関数 M を引数 N に適用すること (すなわち，普通の記法で書くと $M(N)$) を表わす．例えば，x, y, f, g を変数とするとき，

¶A.Church, The culculi of lambda-conversion, *Ann. Math. Studies* 6, Princeton Univ. Press, 1941.

(a) $(\lambda x.x)$, $(\lambda x.(\lambda y.x))$, $(\lambda x.(f(fx)))$, $(\lambda f.(\lambda x.(f(fx))))$ などはいずれもλ式である．

(b) $(\lambda \underline{x}.(\lambda y.\underline{\underline{x}}))$ に現れる 2 つの x は束縛変数であり，$\underline{\underline{x}}$ は $\underline{x}$ によって束縛されている．$\underline{\underline{x}}$ は $(\lambda y.\underline{\underline{x}})$ においては自由である．$((\lambda f.(\lambda x.(f(fx))))(\lambda x.(\underline{f}(\underline{f}x))))$ に現れる 3 つの f は束縛変数であり，2 つの $\underline{f}$ は自由変数である．

このように，λ式の中で同じ名前の束縛変数と自由変数が現れることが可能であるが，束縛変数を別の名前に置き換えた式を元の式と同一視する．こうして，λ式 $M_1, \cdots, M_n$ の間で束縛変数と自由変数が重ならないようにすることができる．このとき，$M_1, \cdots, M_n$ は**変数条件**を満たすという．

(c) $(\lambda x.x)$ は恒等関数 $x \mapsto x$ を表わす．$(\lambda x.(\lambda y.x))$ は x に関数 $y \mapsto x$ を対応させる関数 $x \mapsto (y \mapsto x)$ を表わす．これは 2 変数関数 $(x, y) \mapsto x$ とみなすことができる．$(\lambda x.(f(fx)))$ は，まず x に f を 2 回関数適用し，その結果を x に関して関数抽象して得られたλ式である．すなわち，このλ式は関数 $x \mapsto f(f(x))$ を表わす．$(\lambda f.(\lambda x.(f(fx))))$ は関数 f に関数 $(\lambda x.(f(fx)))$ を対応させる関数 $f \mapsto (\lambda x.(f(fx)))$ を表わし，$((\lambda g.(\lambda x.(g(gx))))(\lambda y.(f(fy))))$ は関数 $x \mapsto f(f(f(f(x))))$ を表わす．

λ式を定義通りに書くと括弧が多くて見にくいため，次のような略記法を用いる:

- $n \geq 1$ のとき，$\lambda x_1 x_2 \cdots x_n.M = (\lambda x_1.(\lambda x_2.(\cdots(\lambda x_n.M)\cdots)))$
- $n \geq 2$ のとき，$M_1 M_2 M_3 \cdots M_n = ((\cdots((M_1 M_2) M_3)\cdots) M_n)$
- (M) と M を同一視する．

L, M, N をλ式，x を変数とする．M, x, N が変数条件を満たすとき，M の中のすべての自由な x を N で置き換えて得られる式を $M[x := N]$ で表わす．

さて，λ記法で表わされた関数を，ある値に対して適用する操作 $\underset{\beta}{\longrightarrow}$ をβ **変換** (β conversion) といい，次の (1),(2) により定義する．L, M, N を任意のλ式とする．

(1) $(\lambda x.M)N \underset{\beta}{\longrightarrow} M[x := N]$.

(2) $M \underset{\beta}{\longrightarrow} N$ ならば $\lambda x.M \underset{\beta}{\longrightarrow} \lambda x.N$, $ML \underset{\beta}{\longrightarrow} NL$, $LM \underset{\beta}{\longrightarrow} LN$.

$\underset{\beta}{\longrightarrow}$ の反射推移閉包を $\overset{*}{\underset{\beta}{\longrightarrow}}$ で表わす．すなわち，$P_1 \underset{\beta}{\longrightarrow} P_2 \underset{\beta}{\longrightarrow} \cdots \underset{\beta}{\longrightarrow} P_n$ を満たす $P = P_1, P_2, \cdots, P_n = Q$ $(n \geq 1)$ が存在するとき $P \overset{*}{\underset{\beta}{\longrightarrow}} Q$ と書く．

それ以上 β 変換が適用できない λ式は**正規形** (β-normal form) であるという．

$n \in \mathbf{N}$ に対して，λ式 $\lambda xy.(\overbrace{x(x(\cdots(x}^{n}\,y)\cdots))$ を $\overline{n}$ で表わす．λ式 S を $S = (\lambda xyz.y(xyz))$ で定義すると，

$$
\begin{aligned}
(S\overline{0}) &\underset{\beta}{\longrightarrow} ((\lambda xyz.y(xyz))\overline{0}) \underset{\beta}{\longrightarrow} (\lambda yz.y(\overline{0}yz)) \\
&= (\lambda yz.y(\lambda xy.y)yz) \underset{\beta}{\longrightarrow} (\lambda yz.y(\lambda y.y)z) \\
&\underset{\beta}{\longrightarrow} (\lambda yz.y(z)) \\
&= (\lambda xy.x(y)) = \overline{1}
\end{aligned}
$$

である．一般に，$(S\overline{n}) \overset{*}{\underset{\beta}{\longrightarrow}} \overline{n+1}$ が任意の $n \in \mathbf{N}$ に対して成り立つ．

任意の自然数 m, n に対して，次のことを示せ．

(a) $(\overline{m}\,\overline{n}) \overset{*}{\underset{\beta}{\longrightarrow}} \overline{n^m}$ (b) $((\lambda xyz.x(yz))\overline{m}\,\overline{n}) \overset{*}{\underset{\beta}{\longrightarrow}} \overline{mn}$

(c) $((\lambda xyzw.((xz)((yz)w))\overline{m}\,\overline{n}) \xrightarrow[\beta]{*} \overline{m+n}$

次の条件を満たす λ式 F が存在するとき，関数 $f : \mathbf{N}^n \to \mathbf{N}$ は **λ定義可能** (λ-definable) であるという: 任意の $x_1, \cdots, x_n, y \in \mathbf{N}$ に対して，

$$f(x_1, \ldots, x_n) = y \iff (F\overline{x_1}\cdots\overline{x_n}) \xrightarrow[\beta]{*} \overline{y}.$$

自然数の上の (部分) 関数 f がλ定義可能であることと f が帰納的 (部分) 関数であることとは同値であることが知られている．

4 この問題では，Herbrand が提唱し Gödel が発展させた HG–計算可能性[‖] について説明する．まず，**項**とは何かを次のように再帰的に定義する：

(1) $x_1, x_2, \cdots$ を変数記号と呼ぶ．変数は項である．

(2) 0 を定数記号と呼ぶ．0 は項である．

(3) t が項ならば $(t)'$ は項である．

(4) f_i^j $(i, j = 0, 1, \cdots)$ を関数記号と呼ぶ．上付き添字の j は何変数関数であるかを表わす．$t_1, \cdots, t_n$ がそれぞれ項で，f_i^n が関数記号ならば $f_i^n(t_1, \cdots, t_n)$ は項である．

自然数 n を表わすために**数値項**と呼ばれる項 $\overline{n}$ を

$$\overline{0} = 0,\ \overline{n+1} = (\overline{n})'$$

で定義する．

s および t を項とするとき，$s = t$ なる形の式を**等式**と呼ぶ．いくつかの等式 $E : s_1 = t_1,\ \cdots,\ s_k = t_k$ を連立させたものを**連立式**と呼ぶ．ただし，各 $s_i = t_i$ $(1 \le i \le k)$ の左辺 s_i は $f_j^n(t_1, \cdots, t_n)$ なる形の項でなければならない．

次の 2 つの変換規則を考える:

R₁ $s = t$ を等式とする．$s = t$ の中のある 1 つの変数記号に注目し，そのすべてに，ある数値項を代入したものを $s' = t'$ とする．このことを $s = t \overset{\mathrm{R_1}}{\Longrightarrow} s' = t'$ と表わす．

R₂ 変数記号を含まない等式 $s = t$ と別の等式 $f_i^n(\overline{m}_1, \cdots, \overline{m}_n) = \overline{m}$ が与えられたとき，t の中の何ヶ所かに現れる項 $f_i^n(\overline{m}_1, \cdots, \overline{m}_n)$ のうちのいくつかを $\overline{m}$ で置き換えたものを t' とする．このことを

$$s = t,\ f_i^n(\overline{m}_1, \cdots, \overline{m}_n) = \overline{m} \overset{\mathrm{R_2}}{\Longrightarrow} s = t'$$

と表わす．

各 e_i を等式とし，等式の列 $e_1,\ e_2,\ \cdots,\ e_n$ を考える．各 e_i $(1 \le i \le n)$ が以下の 3 つの条件のどれかを満たしているとき，$E \vdash e_n$ と書く：

(i) e_i は連立式 E の中の一つの等式である．

(ii) $e_j \overset{\mathrm{R_1}}{\Longrightarrow} e_i$ となる e_j $(j < i)$ が存在する．

(iii) $e_j, e_{j'} \overset{\mathrm{R_2}}{\Longrightarrow} e_i$ となる $e_j, e_{j'}$ $(j < i,\ j' < i)$ が存在する．

(部分) 関数 $f : \mathbf{N}^n \to \mathbf{N}$ が **Herbran-Gödel 計算可能**であるとは，任意の自然数 $k_1, \cdots, k_n, m$ に対して，

$$f(k_1, \cdots, k_n) = m \iff E \vdash f_i^n(\overline{k}_1, \cdots, \overline{k}_n) = \overline{m}$$

となる連立式 E が存在することである．ここで，$f_i^n(x_1, \cdots, x_n)$ は E の最後の等式の左辺である．Herbran-Gödel 計算可能関数と帰納的関数とは一致することが知られている．

[‖] K.Gödel, On undecidable propositions of formal mathematical systems, Princeton, 1934. S.C.Kleene, *Introduction to Metamathematics*, Van Nostrand, 1952.

次の連立式 E は $\mathrm{add}(x, y) = x+y$ を定義することを示せ．すなわち，$E \vdash f_1^2(\overline{x}, \overline{y}) = \overline{x+y}$ がすべての自然数 x, y に対して成り立つ．

$$E \left\{ \begin{array}{l} f_1^2(x_1, 0) = x_1 \\ f_1^2(x_1, (x_2)') = (f_1^2(x_1, x_2))' \end{array} \right.$$

5 TM M が $t(n)$ 時間限定であることの定義を，「長さ n の (受理されるとは限らない) 任意の入力に対して M のステップ数は $t(n)$ 以下である」と変更する．同様に，領域量の定義も変更する．また，$s(n), t(n)$ として任意の部分関数を許すことにする．このような定義の変更が，この章で証明した定理にどのように影響するか考察せよ (3 章の演習問題 3 参照)．

6 命題 2.2 の証明のうち，「帰納的関数が Turing 計算可能である」ことの証明は比較的容易である．すなわち，以下のことを示せ．

(1) 初期関数は Turing 計算可能である．

(2) f と g が Turing 計算可能関数であるならば，これらの合成関数も Turing 計算可能である．

(3) g と h が Turing 計算可能関数であるならば，これらを使って原始帰納法で定義される関数も Turing 計算可能である．

(4) g が Turing 計算可能関数であるならば，これを使って最小化で定義される関数も Turing 計算可能である．

参考までに記すと，関数 f が原始帰納的関数であることと，ある原始帰納的関数 $t(n)$ に対して $t(n)$ 時間限定であるような TM によって f が定義できることとは同値である．

7 $G = (N, \Sigma, P, S)$ を句構造文法とする．

(1) 本来，すべてのプロダクションが $uXv \to uxv$ ($X \in N$, $x \in V^+$, $u, v \in V^*$) という形をしている文法を文脈依存文法といい，本文中で定義されている形の文脈依存文法を**単調文法** (monotonic grammar) という．この 2 つの定義は一致する (生成する言語のクラスが同じである) ことを示せ．

(2) 句構造文法のプロダクションは $x \to y$ ($x \in N^+$, $y \in V^*$) という形に制限できることを示せ．

8 文脈依存言語は帰納的言語であることを示せ．

9 $L \subseteq \Sigma^*$ が帰納的可算な無限集合ならば，L は帰納的な無限部分集合を含むことを示せ．

10 Σ を有限アルファベット，$P = \{(\alpha_1, \beta_1), \cdots, (\alpha_n, \beta_n)\}$ を $\Sigma^+ \times \Sigma^+$ の部分集合 (“規則” の集合) とする．このうちのいくつかを指定して停止規則と呼ぶ．語 $x \in \Sigma^*$ の部分語となるような規則の中で番号が最も小さいものが $\alpha_i \to \beta_i$ であるとき，x の中で最も左に現れる α_i を β_i で書き換えることを $\Rightarrow$ で表わす．$\Rightarrow^*$ を $\Rightarrow$ の反射推移閉包とする．$x \Rightarrow^* y$ かつ y に適用可能な規則が存在しないか，または，$x \Rightarrow^* y$ かつ最後に適用されたのが停止規則であるとき，$x \Rightarrow^* y\rfloor$ と書く．このように $\Rightarrow^*$ が定義されたシステムを**マルコフアルゴリズム** (Markov normal algorithm) という**．

$$\overline{x_1}0\overline{x_2}0 \cdots 0\overline{x_n} \Rightarrow^* \overline{y}\rfloor \Longleftrightarrow \varphi(x_1, x_2, \cdots, x_n) = y$$

が成り立つようなマルコフアルゴリズムが存在するならば，関数 $\varphi : \mathbf{N} \to \mathbf{N}$ は **Markov 計算可能**であるという．Turing 計算可能関数と Markov 計算可能関数は一致することを示せ．

** A.Markov, *The Theory of Algorithms*, Tr. Mat. Inst. Steklov, XLII, 1954.

3
計算量のクラスと基本定理

計算量の理論 (Computational Complexity Theory) の起源は 1960 年代に遡る．M.O.Rabin (1960), A.Cobham (1964), J.Edmonds (1965) らによる初期の論文では，5 章で扱う多項式時間アルゴリズムとか $\mathbf{P} =? \mathbf{NP}$ 問題とかいった重要な概念が非形式的ながらすでに登場しているが，本書で扱うような TM を計算モデルとして時間量・領域量を形式的に定義し，本章 3.1 節・3.2 節および次章 4.1 節・4.2 節で述べる基本定理のほとんどすべてを証明して，今日の計算量理論の隆盛の基を築いたのは J.Hartmanis, P.M.Lewis II, R.E.Stearns である*．

この章では計算量の理論においてとくに重要な概念である「非決定性」という概念をまず導入する．また，時間量や領域量などに関して以後考察する際の基本となる定理を証明する．

3.1 非決定性 TM

アルゴリズムは機械的な操作手順であるから，ある操作につづけて行なう次の操作は一意的に定まっていなければならない．この性質を**決定性** (determinism) と呼ぶ．TM $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ の場合，遷移関数 (動作関数) δ が $(Q - F) \times \Gamma^k$ から $Q \times (\Gamma \times \{-1, 0, 1\})^k$ への部分関数であることがこれに対応しているが，δ の値域を $2^{Q \times (\Gamma \times \{-1,0,1\})^k}$ へと拡張することにより，次の操作に選択の幅を持たせた "非決定性" の TM (アルゴリズム) を考えることができる†．

---

*J.Hartmanis and R.E.Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* 117, pp.285-306, 1965.

J.Hartmanis, P.M.Lewis II, R.E.Stearns, Hierarchies of memory limited computations, *Proc. 6th IEEE Symp. on Switching Circuit Theory and Logical Design*, pp.179-190, 1965.

†非決定性の概念は A.M.Turing が TM という概念を初めて提案した論文の中ですでに導入している: A.M.Turing, On computable numbers with an application to the Entscheidungsproblem, *Proc. London Math. Soc.* 2, pp.230-265, 1936.

非決定性 k テープ TM (nondeterministic k-tape TM, **NTM** と略記する) とは TM と同じ 6 つ組 $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ である．TM と異なるのは遷移関数 δ だけで，δ は $(Q - F) \times \Gamma^k$ から $2^{Q \times (\Gamma \times \{-1,0,1\})^k}$ への部分関数である．I_1, I_2 を ID とするとき，TM では

$$
\begin{aligned}
I_1 &= (q,\ (x_1 a_1 y_1, h_1), \ldots, (x_k a_k y_k, h_k)),\\
\delta(q, a_1, \ldots, a_k) &= (p,\ (b_1, d_1), \ldots, (b_k, d_k)),\\
I_2 &= (p,\ (x_1 b_1 y_1, h_1 + d_1), \ldots, (x_k b_k y_k, h_k + d_k))
\end{aligned}
$$

であるならば $I_1 \vdash_M I_2$ であると定義したが，NTM における $\vdash_M$ の定義では上記の δ のところが

$$\delta(q, a_1, \ldots, a_k) \ni (p,\ (b_1, d_1), \ldots, (b_k, d_k))$$

に変わるだけである．これは，NTM では $\vdash_M$ の各ステップにおいて複数の遷移が選択可能であることを意味する．したがって，NTM における計算は，例えば図 3.1 に示したように，各頂点が 1 つの ID に対応する根付き木 (**計算木** (computation tree) と呼ぶ) になる．この計算木の根は入力 (w としよう) に対する初期 ID であり，$I_1 \vdash_M I_2$ であるとき I_2 は頂点 I_1 の子となっている．葉の中には受理 ID であるものも拒否 ID であるものも，受理 ID でも拒否 ID でもないものもあり得る．また，選択によっては根から葉への道上に同じ ID が繰返し現われる (したがって，計算がループに陥る) ことがあるかもしれない．しかし，これ以後は計算木と呼ぶ場合には頂点数が有限の木に限定する．根から葉への道は NTM M の計算における選択結果の 1 つであり，TM の場合と同様に M の**計算**と呼ぶ．葉が受理 ID (拒否 ID) であれば**受理 (拒否) 計算**と呼ぶのも同様である．葉の中に少なくとも 1 つ受理 ID があるような，w に対する計算木が存在するとき w は M に**受理**されると定義する．M によって受理される Σ 上の語の集合を TM の場合と同様に $L(M)$ で表わす．

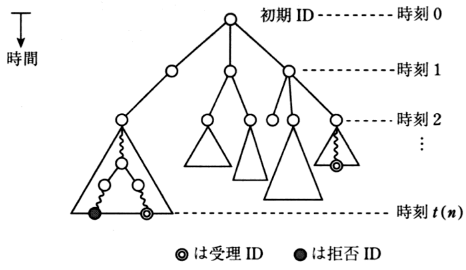


図 **3.1**　NTM における計算 (計算木)

次の条件を満たすとき NTM M は $t(n)$ **時間限定**であるという: どんな入力 w に対しても，もし w が M に受理されるならば，長さが $t(|w|)$ 以下の受理計算が存在する．同様に，次の条件を満たすとき M は $s(n)$ **領域限定**であるという (M の各テープは片方向無限であるとする．両方向無限テープの場合も以下と同様に定義される): どんな入力 w に対しても，もし w が M に受理されるならば，途中の任意の ID $(q,\ (\alpha_1, h_1), \ldots, (\alpha_k, h_k))$ が $h_i \leq s(|w|)$ $(i = 1, \ldots, k)$ を満たしているような受理計算が存在する．したがって，M に受理されない入力 w' に対する計算は時間が $t(|w'|)$ を越えても領域量が $s(|w'|)$ を越えてもかまわないのはもちろんであるが，M に受理される入力 w に対する計算の中にも受理 ID に至らなかったり時間が $t(|w|)$ を越えたり領域量が $s(|w|)$ を越えたりするものがあってもかまわない (演習問題 3 参照)．$t(n)$ 時間限定 NTM によって受理される言語のクラス，$s(n)$ 領域限定 NTM によって受理される言語のクラスをそれぞれ

$$\mathbf{NTIME}(t(n)),\ \mathbf{NSPACE}(s(n))$$

で表わす．前章までに登場した TM は NTM の特別な場合 (計算の各ステップで遷移が一意的に決まっているもの) であり，NTM と明示的に区別する場合には**決定性 TM** (deterministic TM, **DTM** と略記する) と呼ぶ．$t(n)$ 時間限定 DTM によって受理される言語のクラス，$s(n)$ 領域限定 DTM によって受理される言語のクラスをそれぞれ

$$\mathbf{DTIME}(t(n)),\ \mathbf{DSPACE}(s(n))$$

で表わす．これらを**計算量のクラス** (complexity class) という．

例 3.1 (1) $\mathrm{WW}^R = \{ww^R \mid w \in \{a,b\}^*\}$ を受理する 2 テープ NTM M を考えよう．最初のうち M は 1 文字ずつ入力を読みながら第 2 テープにそれをコピーすることをつづけるが，適当な時点で，コピーをつづけることとコピーを止めて以下に述べる比較動作を開始することとを非決定的に選択する．比較動作では，第 1 テープヘッドを右に動かすと同時に第 2 テープヘッドを左に動かしながら，第 1 テープ上の記号と第 2 テープ上の記号が等しいかどうかを判定する．この判定がすべて 'yes' で推移し，第 1 テープヘッドが初めて $\flat$ を読んだときに第 2 テープヘッドも $\flat$ を読んでいれば入力を受理し，それ以外の場合には入力を拒否する．形式的な定義をすると次のようになる: $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, \flat\}, \delta, q_0, \{q_2\})$. $x \in \{a, b\}$ に対して，

$$\begin{aligned}
&\delta(q_0, \flat, \flat) = \{(q_2, (\flat, 0), (\flat, 0))\},\\
&\delta(q_0, x, \flat) = \{(q_0, (x, 1), (x, 1)), (q_1, (x, 1), (x, 0))\},\\
&\delta(q_1, x, x) = \{(q_1, (x, 1), (x, -1))\},\\
&\delta(q_1, \flat, \flat) = \{(q_2, (\flat, 0), (\flat, 0))\}.
\end{aligned}$$

また，擬似プログラムで表現する場合には，次のように**guess 文**を用いる (guess 文では，受理に至るような選択肢を推測して選ぶ):

```
procedure M(w)
begin
    GUESS:
        if 入力ヘッドが読んでいる記号 (x とする) ≠ ♭ then
            x を第 2 テープに書く;
            guess CENTER or NOT of
                case CENTER:
                    入力ヘッドだけを右に 1 こま動かす; goto VERIFY;
                end;
                case NOT:
                    両テープヘッドを右に 1 こま動かす; goto GUESS
                end
            end
        end;
    VERIFY:
        while 入力記号 = 第 2 テープヘッドの読んでいる記号 ≠ ♭ do
            入力ヘッドを右に 1 こま，第 2 テープヘッドを左に 1 こま動かす
        end;
        if 入力記号 = 第 2 テープヘッドの読んでいる記号 = ♭
            then 入力を受理 else 入力を拒否
        end
end
```

入力が ww^R の形をしている場合には，ちょうど前半の w を読み終わったときに guess 文により選択肢 CENTER を選べば，引き続いて VERIFY 部分において後半の w^R が w の鏡像であることが確認されて入力は受理されるから，$ww^R \in L(M)$ である．また，それ以外の場合には受理に至るような選択が不可能なので $L(M) = \mathrm{WW}^R$ である．このように，一般に非決定性の計算は，受理に至る計算を推測し選択しながら進められ (この例では guess 文において ww^R の中央を推測している)，選択した推測が正しければその後の計算で正しかったことが検証されるという方法 (**推測と検証** (guess and verify)) で行なわれる．

明らかに，M は $n+1$ 時間限定，かつ，$n/2+1$ 領域限定であるから，$\mathrm{WW}^R \in \mathbf{NTIME}(n+1) \cap \mathbf{NSPACE}(n/2+1)$ である (問 3.1 参照).

(2) ハミルトン閉路問題が $\mathbf{NTIME}(cn^2) \cap \mathbf{NSPACE}(n)$ に属すこと，すなわち，$L_{Hamilton}$ を受理する cn^2 時間限定かつ n 領域限定の NTM M_H が存在することを示そう (c は適当な定数).

M_H は 3 本のテープを持つ．グラフ G (のコード $\overline{G}$; 例 2.5 (2) 参照) が入力として第 1 テープに与えられると，M_H はまず，ハミルトン閉路を構成するであろうと推測される頂点の列を第 2 テープに書いてゆく (推測ステップ)．もっと具体的に述べよう (以下に登場する $c_1 \sim c_4$ は適当な定数)．1 つの頂点を推

測して選ぶ際に M_H は非決定性の遷移を行なうが，選んだ頂点は一旦第 3 テープに書き込み (かかる時間はたかだか c_1n)，1 つ前に選んだ頂点と隣接しているかどうかを調べる．そのために，M_H は第 2 テープ (末尾に 1 つ前に選んだ頂点が書いてある) と第 3 テープの内容を第 1 テープに書いてある $\overline{G}$ と照合し (このことは，非決定性の動作により c_2n 時間でできる)，隣接していなかった場合には入力を拒否して停止する．隣接していた場合には，第 3 テープの内容を第 2 テープの内容と比較することによって，今回選んだ頂点がそれまでに選んだ頂点のどれとも異なることを決定性動作でチェックする．これにかかる時間はたかだか c_3n である．このチェックに合格した場合だけ第 3 テープの内容を第 2 テープの末尾にコピーして推測ステップをつづけ，それ以外の場合には検証ステップに移る．たかだか n 個の頂点を選べば十分であるから (問: なぜか？ n は G の頂点の個数ではなく，入力の長さである)，以上述べた推測ステップにはたかだか $(c_1+c_2+c_3)n \times n$ 時間使えば十分である．

最後に検証ステップとして，M_H は G の頂点すべてが選ばれて第 2 テープ上に書かれていることと，最後の頂点と最初の頂点が隣接していることを調べる．これにかかる時間はたかだか c_4n である (すべての頂点が選ばれたかどうかは，生成した頂点の個数が G の頂点の個数 ($\overline{G}$ にコード化されて入っている) と等しいことをチェックすればよい．これにかかる時間が $O(n)$ であることは，例 2.4・問 2.4 を参照せよ)．よって，G がハミルトングラフなら，M_H は $\overline{G}$ を $(c_1+c_2+c_3)n^2+c_4n$ 時間で受理することができる (ハミルトングラフでないものを受理しないことはもちろんである)．ここに，$n=|\overline{G}|$．ゆえに，$c=2\max\{c_1+c_2+c_3,\ c_4\}$ とおけば，M_H は cn^2 時間限定である．

一方，第 2 テープに書き込まれる頂点についての条件から，第 2 テープ上で使われる領域はたかだか n である．また，新たに推測した頂点はつねに第 3 テープの左端から書き込むようにすれば，第 3 テープで使う領域もたかだか n とできる．よって，M_H が使う領域量はたかだか n である． □

問 3.1 (1) 例 3.1 (2) の M_H は，ある時点で選んだ頂点が 1 つ前に選んだ頂点と隣接していることと，選んだ頂点がすべて異なることの検証を推測ステップにおいて行なっているが，これを検証ステップにおいて行なうように直せ．直した M_H の時間量と領域量を求めよ．

(2) 例 3.1 (1) の M と同様に，M_H を手続きにより記述せよ．

与えられた問題を解くアルゴリズム (与えられた言語を受理する TM) を設計しようとするとき，"非決定性" の機能があると設計が容易である (読者は，例えば，例 3.1 に挙げた 2 つの言語を受理する DTM を作ってみればこのことが納得できるであろう)．では，非決定性によって本当に計算能力が上がるのであろうか？ 実は，この**決定性問題** (determinacy problem) は計算量理論において

もっとも基本的で重要な問題でありながら，現在まで本質的な解決がほとんど何もなされていない問題の一つである．この問題から派生した数々の有名未解決問題がある．いくつか例を挙げてみよう:

(1) **DL =? NL 問題**: **DSPACE**($\log n$) = **NSPACE**($\log n$) か？ (4.3 節の例 4.2 (3) および 5.3.2 節参照)
(2) **LBA 問題**: **DSPACE**(n) = **NSPACE**(n) か？ (4.3 節の例 4.2 (2) および 4 章の演習問題 4 参照)
(3) **P =? NP 問題**: $\bigcup_{k\geq 0}$ **DTIME**(n^k) = $\bigcup_{k\geq 0}$ **NTIME**(n^k) か？ (この問題については 5.2 節で詳細に述べる)

チューリングマシン (NTM でも DTM でもよい) M_1 が与えられたとき，M_1 を**模倣** (simulate) するチュリングマシンとは $L(M_1) = L(M_2)$ なる M_2 のことをいう．時間量に関しては，DTM による NTM の模倣法として次のような自明の方法 (NTM の計算木上の可能な道すべてをなぞってみる方法) しか知られていない．

定理 3.1 M を $f(n)$ 時間限定 NTM とすると，M を模倣する $c^{f(n)}$ 時間限定 DTM M' が存在する．

証明 M のテープ本数を k，状態の個数を s，テープ記号の個数を t とする．M が取り得る ID の個数はたかだか「取り得る状態の個数」×「k 個のヘッド位置に関する場合の数」×「k 本のテープの内容に関する場合の数」である．M は $f(n)$ 時間限定だから，どのテープ上でもたかだか $f(n)+1$ 領域しか使わないから，この値はたかだか $s(f(n)+1)^k t^{kf(n)}$ である．ここで，n は入力の長さ．$d = s(t+1)^{3k}$ とおくと，任意の $n \geq 1$ に対して

$$s(f(n)+1)^k t^{kf(n)} \leq d^{f(n)}$$

である．

M がある ID から次の ID へ遷移するとき選択できる可能性がたかだか r 通りであるとする．M' は $k+r+1$ 本のテープを持つ．入力 w ($|w| = n$) が与えられたとき M' は，w に対する初期 ID から M が到達可能な ID すべてを重複しないように第 $k+1$ テープ上に生成する．この生成は，図 3.1 に示した計算木を幅優先探索 (1 章の演習問題 7 参照) するような順序で行なう．

すでに ID $I_1, \ldots, I_i$ が生成済みで，そのうちの $I_1, \ldots, I_{j-1}$ の子のすべてが生成済みとする (図 3.2 参照)．ID 間には境界を表わすます目があるが，とくに I_{j-1} の右側の境界ます目に印を付けておく．I_j のすべての子を次のように生成する．M' は第 $k+1$ テープヘッドを I_j の左端に移し，I_j の子となり得る

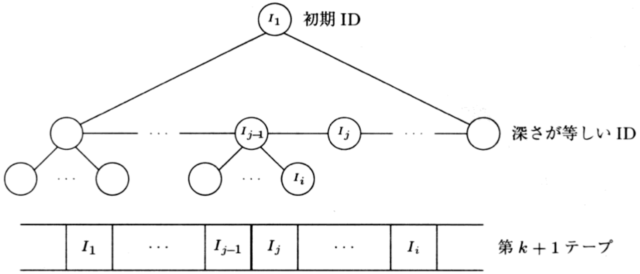


図 3.2 M の計算木と ID の生成

ID を 1 つずつ第 $k+2 \sim k+r+1$ テープ上に生成する (r 個より少ないこともある)．次に，テープ $k+1 \sim k+r+1$ のヘッドを左端に戻し，生成された子 ID の中に $I_1 \sim I_i$ に等しいものがないかどうか調べる．$I_1 \sim I_i$ のどれとも等しくなかったものだけを I_i の右にコピーし，I_j の右側の境界ます目に印を付ける．以上の操作は，ID のリスト $I_1 \sim I_i$ の長さに比例する時間で行なうことができる．1 つの ID は長さが $O(1+k(f(n)+1))$ の文字列 (すなわち，状態を表わす記号と，テープの内容を表わす k 個の文字列 (各文字列は $\alpha \uparrow \beta$ なる形で，$\uparrow$ はヘッドの位置を表わす記号) を連接したもの) で表わすことができる．したがって，最終的にできるリストの長さ ℓ は $O(d^{f(n)}(1+k(f(n)+1)))$ であり，M' が使う時間は $O\left(\sum_{i=1}^{\ell} i\right) = O(\ell^2)$ である．$\ell^2 \leq c^{f(n)}$ となる定数 c が存在する．

上記の生成の途中で M の受理 ID が生成されたら M' は入力を受理する．受理 ID が生成されないまま M の全 ID の生成が終了したら M' は入力を拒否する． □

問 3.2 次は定理 3.1 の証明になるか？(3.2.3 節参照)
M' は $k+2$ テープとし，第 $k+1$ テープ上に $\{1,\ldots,r\}^+$ の語を系統的に順次生成する．それらの語は M の計算木 T の根から葉への 1 つの道を表わす (T の道にならないものもある)．M' は $\{1,\ldots,r\}^+$ の語を 1 つ生成するたびに，第 $k+2$ テープに入力をコピーし，それを使って第 $k+1$ テープ上の語に対応するように M の動作を模倣する．M が入力を受理したら M' も入力を受理してそこで停止する．M が入力を拒否した場合や，第 $k+1$ テープ上に生成された語が T の道 (M の動作) としてあり得ないものであった場合は，次の語の生成に移る．長さ $f(n)$ 以下の語をすべて生成し終わり，そのどの場合にも，対応する M の動作が受理に至らなかったとき，M' は入力を拒否する．

領域量に関しては，定理 3.1 よりもっとよい結果を証明することができる (定理 4.2 参照)．

3.2　基本定理

3.2.1　計算量の線形縮小

すでに 1.7 節でも述べたが，時間量にしても領域量にしても定数倍の違いは本質的なものではない．実際，TM を計算モデルとする場合，計算のスピードは何倍でも加速することができるし (線形加速定理，Linear Speed-up Theorem)，計算に使うスペースは何分の 1 にでも圧縮することができること (線形領域圧縮定理，Space Compression Theorem) をこの節で証明する．まず，線形領域圧縮定理の証明の準備として次の定理を証明する．

定理 3.2 L が $s(n)$ 領域限定の k 作業テープオフライン DTM (NTM) M によって受理されるなら，L を受理する $s(n)$ 領域限定の 1 作業テープオフライン DTM (NTM) M' が存在する．

証明 M' は作業テープを $2k$ トラックに分け，第 $2i-1$ トラックには M の第 i 作業テープの内容を，第 $2i$ トラックには M の第 i 作業テープのヘッドがその位置にあるか否か (記号 $\uparrow, \flat$ で表わす) を書き込む．M' において空白記号が書かれているます目は，M の k 本のテープのどれにおいても，対応するます目位置がまだ未使用であることを表わしている (図 3.3 参照)．

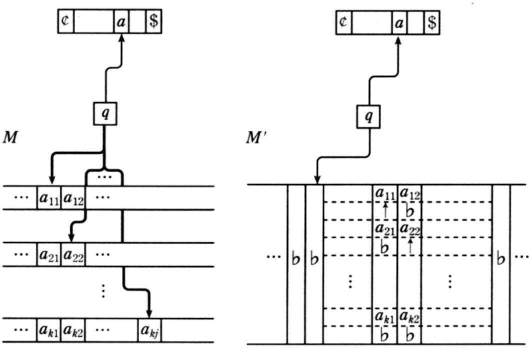


図 **3.3**　k テープを $2k$ トラックで表わす

M は DTM とし (NTM の場合も同様である)，δ を M の遷移関数とする．M が $\delta(q, a, a_{1j_1}, \ldots, a_{kj_k}) = (p, d, (b_{1j_1}, d_1), \ldots, (b_{kj_k}, d_k))$ によって遷移するとき (p, q は状態，a は入力記号，a_{ij_i}，b_{ij_i} はテープ記号，d, $d_i \in \{-1, 0, 1\}$ は入力ヘッド・第 i 作業ヘッドの移動方向，をそれぞれ表わす)，M' はこの 1

ステップを次のように模倣する．模倣開始時に M' の作業テープヘッドはテープの左端 ($\flat$ が書いてあるこま) にある．

1. 状態 q を有限制御部に記憶する．
2. 作業テープヘッドを右に動かしながら，M のテープヘッドがある位置の記号 (と第何テープ上か) を優先制御部に記憶する．作業テープの右端 (最初の $\flat$ に出会う位置) に到達したとき，$(a_{1j_1}, \ldots, a_{kj_k})$ が有限制御部に記憶されている．
3. $(p, d, (b_{1j_1}, d_1), \ldots, (b_{kj_k}, d_k))$ を有限制御部に作る．
4. 作業テープヘッドを左に動かしながら，M のヘッドがある位置の a_{ij_i} を b_{ij_i} に書き換えると同時に d_i に従って，ヘッド位置を表わす記号 $\uparrow$ も書き換える．
5. 作業テープの左端に到達したら，状態を p に変え，入力ヘッドを d に従って動かす．M が入力を受理したら M' も入力を受理する．

明らかに，M' は $s(n)$ 領域限定である． □

系 3.1 L が $t(n)$ 時間限定の k 作業テープ DTM (NTM) M によって受理されるなら，L を受理する $t(n)^2$ 時間限定の 1 作業テープ DTM (NTM) M' が存在する．

証明 定理 3.2 の証明とまったく同様にやればよい．M が $t(n)$ 時間限定なら M' が M の 1 ステップを模倣するのにかかるステップ数は $O(t(n))$ であるから，$t(n)$ ステップを模倣するのにかかるステップ数は $O(t(n)^2)$ である．後述するように (定理 3.4)，$O()$ をはずすことができる． □

補題 3.1 $s : \mathbf{N} \to \mathbf{N}$ を任意の関数，m を任意の正整数とする．L が $s(n)$ 領域限定の 1 作業テープオフライン DTM (NTM) M によって受理されるなら，L を受理する $\lceil s(n)/m \rceil$ 領域限定の 1 作業テープオフライン DTM (NTM) M' が存在する．

証明 M' は M の連続する m 個のます目の内容を 1 つのテープ記号で表わす．すなわち，M のテープアルファベットが Γ のとき，M' のテープアルファベットを Γ^m とする．M の作業テープヘッドが $x \in \Gamma$ を指しているとき，M' の作業テープヘッドは Γ^m の元 $[\cdots x \cdots]$ (M の作業テープヘッドが指しているます目を中間に含んでいる連続 m 個のます目を表わす) を指すように動作する．x がこの記号の中の何番目の文字であるかは M' の有限制御部に記憶する．

すなわち，M' は $[q,i]$ という形の状態を持つ．q はそのときの M の状態，i は x の位置 $(1 \le i \le m)$．例えば，M が DTM で，M の遷移が

$\delta(q,a,x) = (p,d,(b,d_1))$　(d, d_1 は入力ヘッド・作業ヘッドの移動方向)

であるとき，

$1 < i < m$ のとき，$\delta'([q,i],a,[\cdots x \cdots]) = ([p,i+d_1],d,([\cdots b \cdots],0))$,

$d_1 = 1$ のとき，$\delta'([q,m],a,[\cdots x]) = ([p,1],d,([\cdots b],1))$,

$d_1 = -1$ のとき，$\delta'([q,1],a,[x \cdots]) = ([p,m],d,([b \cdots],-1))$

とする．明らかに，M' は $\lceil s(n)/m \rceil$ 領域限定である．　□

定理 3.2，補題 3.1より，次の定理を得る[‡]．

定理 3.3 (線形領域圧縮定理) 任意の関数 $s: \mathbf{N} \to \mathbf{N}$, 任意の実数 $c > 0$ に対し，$\mathbf{DSPACE}(cs(n))$=$\mathbf{DSPACE}(s(n))$, $\mathbf{NSPACE}(cs(n))$=$\mathbf{NSPACE}(s(n))$ である．

今後，領域量として $\log n$, $\log f(n)$ などがしばしば登場するが，定理 3.3 により，対数の底は何でもよいことが分かる．本書では，$\log 0 = 0$ と定義され，例えば $\log n$ は $\max\{1, \lceil \log_2 n \rceil\}$ の省略形である (p.42 の脚注参照)．

補題 3.2 $k \ge 2$, $m > 0$ を任意の整数とする．L が $t(n)$ 時間限定の k テープ DTM (NTM) M によって受理されるなら，L を受理する $n + 1 + \lceil n/m \rceil + 3\lceil t(n)/m \rceil$ 時間限定の k テープ DTM (NTM) M' が存在する．

証明 補題 3.1 と同様に (Γ, δ の意味も同じとする)，M' は M の連続する m 個のます目の内容を 1 つのテープ記号 ($\in \Gamma^m$) で表わす．さらに，M' は M のヘッドがある付近の $3m$ 個のます目の内容とヘッド位置を有限制御部に記憶しておく．すなわち，M の第 i テープが図 3.4 の状況にあるとき，M' の有限制御部には $(\ldots, [\alpha\beta\gamma, \ell], \ldots)$ が記憶される．ここに，$\alpha, \beta, \gamma \in \Gamma^m$, $1 \le \ell \le 3m$ であり，ℓ は M のテープヘッドが $\alpha\beta\gamma$ の左から何番目の記号を指しているか表わす．

M' は次のように M の動作を模倣する．

1. 入力文字列を m 文字ずつまとめて 1 つの記号 (Γ^m の元) にして第 2 テープに書き出す．同時に，入力テープの内容は消去する ($\flat$ を書き込む)．入力の長さを n とすると，この部分の操作に要するステップ数は $n+1$ である．
2. 第 2 テープのヘッドをテープの左端に戻す．これにかかるステップ数は $\lceil n/m \rceil$ である．第 1 テープ (入力テープ) のヘッドは元に戻す必要はな

[‡]他書では，単に「領域圧縮定理」と呼んでいることが多い．

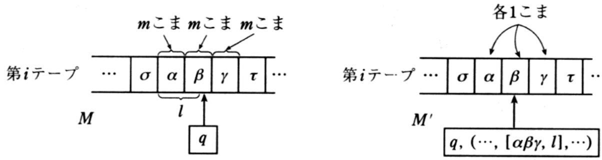


図 **3.4**　M と M' の第 i テープと有限制御部

く，これ以降，M' では第1テープと第2テープの役割を M のそれと逆にする．

3. 以後，M の m ステップを以下のようにたかだか3ステップで模倣する．第 i テープだけに注目して述べるが，他のテープについても同時に同様に行なう．

   図3.4の状況にあるとする．M が次の m ステップの動作で $\alpha\beta\gamma$ を $\alpha'\beta'\gamma'$ に書き換え $(\alpha', \beta', \gamma' \in \Gamma^m)$，状態を p に変えたとする．また，そのときテープヘッドは $\alpha'\beta'\gamma'$ の左から ℓ' 番目の文字を指していたとする．そのような遷移が可能かどうかは δ から計算できるので，M' は状態を $p, (\ldots, [\alpha'\beta'\gamma', \ell'], \ldots)$ に変えることができる (実際には，この時点では状態を変えない)．M の m ステップの動作しか考慮していないので，$\alpha\beta$ の部分か $\beta\gamma$ の部分しか変わっていないことに注意する．

   (a) $\ell' \le m$ のとき，M' は第 i ヘッドが現在指しているます目とその左のます目にそれぞれ β', α' を書き込み，さらに左のます目の σ を読んで，状態を $p, (\ldots, [\sigma\alpha'\beta', m+\ell'], \ldots)$ に変え，ヘッドを元の位置の1つ左に移す．

   (b) $m+1 \le \ell' \le 2m$ のとき，$\alpha = \alpha'$ か $\gamma = \gamma'$ である．前者の場合，M' は現在のます目とその右のます目にそれぞれ β', γ' を書き込み，状態を $p, (\ldots, [\alpha'\beta'\gamma', \ell'], \ldots)$ に変える．ヘッド位置は元の位置と同じ．$\gamma = \gamma'$ の場合も同様である．

   (c) $2m+1 \le \ell'$ のとき，M' は現在のます目とその右のます目にそれぞれ β', γ' を書き込み，さらにもう1つ右のます目の τ を読んで，状態を $p, (\ldots, [\beta'\gamma'\tau, \ell'-m], \ldots)$ に変え，ヘッド位置を元の位置の1つ右に移す．

以上の模倣に M' が費やすステップ数は総計でたかだか

$$n + 1 + \lceil n/m \rceil + 3\lceil t(n)/m \rceil$$

である．　□

定理 3.4 (線形加速定理) 実数 $d > 1$ が存在し，$t : \mathbf{N} \to \mathbf{N}$ は $t(n) \geq dn$ を満たす関数とする．このとき，任意の実数 $c > 0$ に対し，$\mathbf{DTIME}(ct(n)) = \mathbf{DTIME}(t(n))$, $\mathbf{NTIME}(ct(n)) = \mathbf{NTIME}(t(n))$ である．

証明 M を $ct(n)$ 時間限定 DTM (NTM)，$L = L(M)$ とする．補題 3.2 より，任意の整数 $m \geq 1$ に対し，L を受理する $n + 1 + \lceil n/m \rceil + 3\lceil t(n)/m \rceil$ 時間限定 DTM (NTM) M' が存在する．仮定より $t(n)/d \geq n$．また，任意の実数 x に対して $\lceil x \rceil < x + 1$, $n \geq 5m$ なら $t(n) \geq dn \geq 5dm$ であるから，

$$
\begin{aligned}
n + 1 + \lceil n/m \rceil + 3\lceil t(n)/m \rceil &\leq n + n/m + 3t(n)/m + 5 \\
&\leq \left(\tfrac{1}{d} + \tfrac{1}{dm} + \tfrac{3}{m} + \tfrac{1}{dm}\right) t(n)
\end{aligned}
$$

である．よって，$1/d + 2/(dm) + 3/m \leq 1$ となるように m を選べば，長さが $5m$ 以上の入力に対して M' は $t(n)$ 時間限定となる．$d > 1$ であるから，このような正整数 m は存在する．

一方，長さが $5m$ 未満の入力すべてに対して，それが L に属すか否かあらかじめ計算して (M' がではなく，人間が)，有限制御部に記憶しておく．これらの入力に対しては，M' は入力を読み切った時点で停止するようにする．以上で，任意の入力に対して M' の時間量は $t(n)$ 以下となる． □

3.2.2 テープ本数の削減

すでに定理 3.2 において，DTM でも NTM でも，領域量を変えることなく k テープを 1 テープ (オフライン型の場合は入力テープ + 1 作業テープ) にできることを証明した．これはベストの結果である．また，系 3.1 において，$t(n)$ 時間限定 k テープ DTM (NTM) を模倣する $t(n)^2$ 時間限定 1 テープ DTM (NTM) が存在することを示した．1 テープの場合には $t(n)^2$ 時間の模倣はこれ以上改良できないことを次節で示す．しかし，2 本のテープがあると，もっと効率の良い模倣が可能である．

定理 3.5 $k \geq 2$ とする．L が $t(n)$ 時間限定の k テープ DTM (NTM) M によって受理されるなら，L を受理する $t(n) \log t(n)$ 時間限定の 2 テープ DTM (NTM) M' が存在する．

証明 M' の第 1 テープを $2k$ トラックに分け，M の 1 テープごとに 2 トラックを割り当てる．M' の第 2 テープは，第 1 テープ上でデータを転送するときに作業用に使う．以下では，M の 1 本のテープに注目し，それを M' の第 1 テープの 2 トラックを使って模倣する方法を述べる．M' のトラックを，図 3.5 に

示したようにブロック $\ldots, B_{-2}, B_{-1}, B_0, B_1, B_2, \ldots$ に分割する．ブロック B_i の長さは $2^{|i|-1}$ (ただし，B_0 の長さは 1) である．これらのトラックやブロックは必要が生じたとき (M' がブロック化されていないます目を初めて読んだとき) に作られる．ブロックとブロックの間には境界を示すマーカーを置く．

M が最初に読んだます目を a_0 とし，その左右のます目を $\ldots, a_{-2}, a_{-1}, a_0, a_1, a_2, \ldots$ とする (以下では，a_i はそのます目に書かれている文字自身も表わす)．図 3.5 では，それらの文字が 2 つのトラックに分散して置かれている．ただし，その配置は次の条件を満たしていなければならないものとする．トラック上で a_i が書いてないます目には，そのことを表わすために空白文字 $\flat$ が書いてあるものとする (実際は，M' は $2k$ 個の文字の組を 1 つのテープ記号としており，a_i や $\flat$ はそんなテープ記号の 1 つの成分である)．

1. ブロック B_0 の上のトラックはつねに空で，下のトラックはつねに空でない．
2. どのブロックの上下どちらのトラックも，ます目すべてが $\flat$ である (このとき，そのブロックのそのトラックは空であるという) か，すべてが $\flat$ でない．空でないトラックには，M のテープの連続するます目の内容がその順に書かれている．
3. B_i の上のトラックの内容を $\overline{B_i}$ で，下のトラックの内容を $\underline{B_i}$ で表わす (トラックが空のとき，これらは空語 ε とする) と，

$$\cdots \underline{B_{-2}}\overline{B_{-2}}\underline{B_{-1}}\overline{B_{-1}}\underline{B_0}\overline{B_1}\underline{B_1}\overline{B_2}\underline{B_2}\cdots$$

   は M のテープ内容である．
4. B_i $(i \neq 0)$ の上下両トラックがともに空でないなら，B_{-i} の上下両トラックはともに空である．
5. B_i $(i \neq 0)$ の上下両トラックがともに空なら，B_{-i} の上下両トラックはともに空でない．

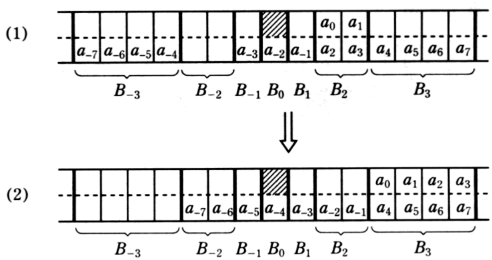


図 3.5　M' における B_t 操作

M のヘッドが a_i にあるとき M' はブロック B_0 に a_i があるように M を模倣する．M が a_i を書き換えるとき，M' はブロック B_0 において a_i を書き換える．このため，M がヘッドを右 (左) に移動させるとき，M' はデータを左 (右) に移動させる．M' は M の 1 ステップを次のように模倣する．

いま，M は a_i を b_i に書き換え，ヘッドを右に移動させたとする．そのとき，M' のブロック B_0 には a_i が書いてある．M' は a_i を b_i に書き換えた後，以下のように動作して a_{i-1} をブロック B_0 に移す．

1. まず，a_i を b_i に書き換える．
2. ヘッドを右 (M のヘッドが動く方向と逆) に動かし，上下どちらかのトラックが空のブロックを探す．最初に見つかったそれを B_t とする．ここでは仮に，B_t の上のトラックが空だったとする (下のトラックが空の場合も同様)．このとき同時に，$B_0, B_1, \ldots, B_{t-1}$ の内容を第 2 テープにコピーする．これにかかる時間は $O(|B_t|)$ である．ここに，$|B_t|$ はブロック B_t の長さを表わす．
3. ヘッドを左に動かしながら，第 2 テープに作ったコピーを B_t の上のトラックと，$B_{t-1} \sim B_1$ の下のトラックに書き込む．この操作にかかる時間は $O(|B_t|)$ である．
4. ブロック B_{-t} の位置を探す．テープ 2 に $B_0, B_1, \ldots, B_{t-1}$ がコピーしてあるので，それを使えば B_{-t} の位置は $O(|B_t|)$ 時間で求められる．
5. B_{-t} の上のトラックの内容 (上のトラックが空の場合は下のトラックの内容) をテープ 2 にコピーする．かかる時間は $O(|B_t|)$ である．
6. 条件 4 より，$B_{-(t-1)}, \ldots, B_{-1}$ は上下のトラックが空である．したがって，テープ 2 にコピーした B_{-t} の内容を $B_{-(t-1)}, \ldots, B_{-1}, B_0$ の下のトラックにコピーするだけのスペースがある．この操作にかかる時間もやはり $O(|B_t|)$ である．この操作の結果，a_{i-1} はブロック B_0 の下のトラックに移ることに注意する．

上記の操作 (**B_t 操作**と呼ぶことにする) にかかる時間は総計で $O(|B_t|)$ である．この操作を行なった結果，各ブロックが上記の必要条件を再び満たしていることは容易にわかる．図 3.5 に B_3 操作の例を示した．

M がヘッドを左に動かした場合の模倣も上記と同様である．また，M がヘッドを動かさなかった場合は，M' は単に B_0 の内容を書き換えるだけでよい．

さて，M' は M の 2^{t-1} ステップにつきたかだか 1 回の B_t 操作をすれば十分である (2 つの B_t 操作の中間は B_j $(j < t)$ 操作の連続である)．なぜなら，1 回 B_t 操作を行なうと $B_1, \ldots, B_{t-1}$ ブロックそれぞれの上下どちらかのトラックは空になるから，このすべてが埋まる (そのとき初めて B_t 操作が適

用可能になる) までに M は少なくとも 2^{t-1} ステップ動く必要がある．また，M' が初めて B_t 操作をするのは，M が少なくとも 2^{t-1} ステップ動いた後である．よって，$2^{t-1} \leq t(n)$ すなわち $t \leq \log_2 t(n) + 1$ なる t についてのみ B_t 操作が行なわれる．ゆえに，1 回の B_t 操作にかかる時間を $c2^t$ とすると (c は定数)，M' の時間量 $t'(n)$ は 1 本のテープにつき

$$t'(n) = \sum_{t=1}^{log_2 t(n)+1} c2^t \frac{t(n)}{2^{t-1}} = O(t(n) \log t(n))$$

である．線形加速定理により，M' を定理の条件を満たすものとすることができる． □

定理 3.5 はもちろん NTM に対しても成り立つが，NTM の場合には 2 テープあれば時間量を変えない模倣が可能である．

定理 3.6 $k \geq 2$ とする．L が $t(n)$ 時間限定の k テープ NTM M によって受理されるなら，L を受理する $t(n)$ 時間限定の 2 テープ NTM M' が存在する．

証明 $M = (Q, \Sigma, \Gamma, \delta, q_1, F)$ とする．$\delta(q,\ a_1,\ \ldots,\ a_k) \ni (p,\ (b_1, d_1),\ \ldots,\ (b_k, d_k))$ のとき，$(q,\ a_1,\ \ldots,\ a_k,\ p,\ (b_1, d_1),\ \ldots,\ (b_k, d_k))$ を M の**動作命令**と呼ぶ．M も M' も第 1 テープが入力テープであるとする．

M' は次のように M を模倣する．

1. まず，M' は入力とは無関係に，テープ 2 に動作命令の列 $\sigma_1\sigma_2\cdots\sigma_\ell$ を非決定的に推測して書き出す．ただし，$\sigma_j = (q_j, a_{j1}, \ldots, a_{jk}, p_j, (b_{j1}, d_{j1}), \ldots, (b_{jk}, d_{jk}))$ とするとき，q_1 は M の初期状態，$p_j = q_{j+1}$ $(1 \leq j < \ell)$，p_ℓ は M の受理状態であるとする．
2. $\sigma_1\sigma_2\cdots\sigma_\ell$ が M の正しい計算になっているかどうかを第 1 ～ k テープごとにチェックする．すなわち，各 $i = 1 \sim k$ に対して次のことを $j = 1 \sim \ell$ について行なう:
   (a) 第 i テープヘッドの読んでいる文字が a_{ji} でないなら入力を拒否して停止する．
   (b) そうでないとき，a_{ji} を b_{ji} に書き換え，第 i テープヘッドを d_{ji} だけ動かす．

   このチェックは第 1 テープを第 i テープの代わりに使って行なう (ただし，2 トラックに分け，入力を第 1 トラックに保存する)．
3. すべての $1 \leq i \leq k$ に対して上記 2 のチェックが ok だったら入力を受理する．

M' が上記のことを行なうのにかかる時間は $O(\ell)$ である．さて，x が M に受理されるなら，長さが $\ell \le t(|x|)$ の，x の受理計算が存在する．この受理計算に対応する動作命令の列を $\sigma_1\sigma_2\cdots\sigma_\ell$ とすると，M' は上記のステップ 1 において $\sigma_1\sigma_2\cdots\sigma_\ell$ を予測し，$O(\ell)$ 時間でそれが正しい計算であることを判定して x を受理する．よって，M' は $O(t(n))$ 時間限定である．線形加速定理により，M' を模倣する $t(n)$ 時間限定 NTM が存在する． □

3.2.3 構成可能関数

問 3.2 に述べられた証明はそのままでは正しくないが，$f(n)$ に時間構成可能と呼ばれる条件を付けると正しい．なぜなら，その場合には $f(n)$ をあらかじめ適当なテープ上に tally 表現 (1 進数) で作っておくことができるから，それを使うと長さ $t(n)$ 以下の語をすべて生成し終わったかどうかを $O(c^{f(n)})$ 時間で判定でき，M' は $O(c^{f(n)})$ 時間で停止することができるからである (そうでない場合には，M が拒否する入力に対して M' は永久に $\{1,\ldots,r\}^+$ の語を生成しつづける)．この例からもわかるように，時間量や領域量に関して "構成可能" であるという概念が重要な役割を果たす．

次の条件を満たす $f(n)$ 時間限定 DTM M が存在するとき，関数 $f(n) : \mathbf{N} \to \mathbf{N}$ は**時間構成可能** (time constructible) であるという: 任意の $n \in \mathbf{N}$ と長さ n の任意の入力 w に対し，M の w に対する計算はちょうど $f(n)$ ステップで停止する．同様に，任意の $n \in \mathbf{N}$ と長さ n の任意の入力 w に対し，w に対する計算が，ある 1 つの決められたテープ上でちょうど $f(n)$ 領域を使うような $f(n)$ 領域限定 DTM M が存在するとき，$f(n)$ は**領域構成可能** (space constructible) であるという (演習問題 4 参照)．

定義より，時間構成可能なら領域構成可能である．n^k，$\log n$，c^n，$n!$ など，時間量や領域量として現れる関数のほとんどは時間構成可能であり，領域構成可能でもある．

例 3.2 (1) n^2 が時間構成可能であることを示そう．次のように動作する DTM を考える．まず，入力の先頭 1 文字を読み (入力ヘッドは右へ動かす)，記号 '1' を第 2 テープに書く (第 2 テープヘッドは動かさない)．以後，入力ヘッドを右に 1 動かすごとに次の操作を行なって停止する (入力が空語のときは何もしないで停止する): $\flat$ に出会うまで第 2 テープヘッドを左に動かし，次いで，'1' が続く限り右に動かし，'1' の右隣りの $\flat$ を '1' に書き換える (ヘッドは右端の '1' の上に残す)．この操作の冒頭で，入力ヘッドも右に 1 動かし，入力が尽きたかどうかをチェックしておく (1 ステップ余計に使うわけではなく，第 2 テープ

ヘッドを左に動かすのと同時に行なう). 操作開始前に '1' が i 個あった場合, この操作にちょうど $2i+1$ ステップかかる. したがって, 停止するまでにかかる時間は $\displaystyle\sum_{i=0}^{n-1}(2i+1) = n^2$ である.

(2) $\lfloor \log_2 n \rfloor$ が領域構成可能であることを示そう. $n > 0$ のとき, n の 2 進数表現の桁数は $\lfloor \log_2 n \rfloor + 1$ であることに注意する. 入力を 1 文字ずつ読みながら, 作業テープに 2 進数として 1 を足していく DTM M を作ることができる (例 2.4 参照). M の領域量は $\lfloor \log_2 n \rfloor + 2$ であるが, 線形領域圧縮定理と同様な方法により, 使用する領域量をちょうど $\lfloor \log_2 n \rfloor$ とすることができる.

(3) $f_1(n), f_2(n)$ がそれぞれ領域構成可能関数ならば, $f_1(f_2(n)), f_1(n)+f_2(n)$, $f_1(n)f_2(n)$, $2^{f_1(n)}$ もそれぞれ領域構成可能関数である. 例えば, $f_1(n)f_2(n)$ を考えよう. $f_1(n), f_2(n)$ がそれぞれ DTM M_1, M_2 によって領域構成可能であるとする. 次のように動作する DTM M を考える.

1. M は, まず第 1 テープの内容 (入力) を第 2 テープにコピーする.
2. 次いで, 第 2 テープを入力テープ代りにして M_1 を模倣しながら第 3 テープの $f_1(n)$ 領域に印を付ける (このとき, 第 1, 5 テープは使わない).
3. つづいて M は第 1 テープの内容 (入力) を再び第 2 テープにコピーし, 第 2 テープを入力テープ代りにして M_2 を模倣しながら第 4 テープの $f_2(n)$ 領域に印を付ける (このとき, 第 1, 3, 5 テープは使わない).
4. 第 3, 4 テープのヘッドを左端に戻す.
5. 第 3 テープヘッドを 1 だけ右に進めるごとに, 第 4 テープヘッドを端から端まで移動する. これを, 第 3 テープヘッドが右端に到達するまで繰り返す. この過程の 1 ステップごとに第 5 テープの 1 こまを使う.

M が第 5 テープ上でちょうど $f_1(n)f_2(n)$ 領域を使うことは明らかである (入力は第 1 テープ上に保存されている). □

読者は上例の (2),(3) を時間構成の場合について証明してみるとよい. 簡単ではないことが納得できるであろう. 実は, 上記の時間 (領域) 構成可能性の定義は時間 (領域) 構成可能であるという性質を使うときにはきわめて都合のよいものであるが, 与えられた関数が時間 (領域) 構成可能であることを, この定義に従って証明するのはむずかしい場合が多い. そんなときには, 次の定理が役立つ. まず, 補題を証明する.

補題 3.3 $f_1(n) + f_2(n)$ および $f_2(n)$ を時間構成可能関数とする. $f_1(n) \geq cf_2(n) + (1+c)n$ を満たす実数 $c > 0$ が存在するならば, $f_1(n)$ も時間構成可能関数である.

証明 $f_1(n)+f_2(n)$, $f_2(n)$ を構成する DTM をそれぞれ M_{12}, M_2 とする．DTM M_1 を次のように構成する．r を 3 より大きい整数とする (具体的な値は後で決める).

段階 1　M_1 は入力 (長さ n とする) の連続する r 文字ずつを 1 記号に圧縮して第 1 テープに書き込む．同時に，$r-3$ 文字ずつを 1 記号に圧縮したものを第 2，第 3 テープに書き込む．この圧縮操作にかかる時間はちょうど n である．この操作の過程で M_1 は $r-3$ を法 (mod) としてステップ数のカウントを行ない，操作終了時にさらに $n \bmod (r-3)$ ステップだけ無駄な動作を行なう．こうすることによって，段階 1 にかかる時間は $r-3$ の整数倍，すなわち $(r-3)\lceil n/(r-3)\rceil$ となる．

段階 2　M_1 は補題 3.2 と同様な方法により，第 1 テープを入力として M_{12} を模倣すると同時に，第 2 テープを入力として M_2 を模倣する．すなわち，M_1 はちょうど 3 ステップかけて，M_{12} の r ステップと M_2 の $r-3$ ステップを同時に模倣する．M_2 の模倣が先に終了するが，そのとき M_1 は M_{12} の模倣を一旦中止する．M_2 の実行時間は $r-3$ の倍数とは限らない．そこで，この過程においても段階 1 と同様に $r-3$ を法としてステップ数のカウントを模倣と並行して行ない，必要なら無駄な動作をさらに行なって，段階 2 の実行時間が $r-3$ の倍数となるようにする．この無駄なステップ数は $t=(r-3)\lceil f_2(n)/(r-3)\rceil - f_2(n)$ であり，段階 2 にかかる時間は $3\lceil f_2(n)/(r-3)\rceil + t = r\lceil f_2(n)/(r-3)\rceil - f_2(n)$ である．段階 2 の終了時点において M_1 は，M_{12} の動作の最初から $r\lceil f_2(n)/(r-3)\rceil$ ステップ分の模倣が済んでいる．

段階 3　M_1 は M_{12} の模倣を再開し，ちょうど $3(\lceil n/(r-3)\rceil+1)$ ステップ使って M_{12} の $r(\lceil n/(r-3)\rceil+1)$ ステップ分を模倣して停止する．このときに必要な値 $\lceil n/(r-3)\rceil$ は第 3 テープの文字列の長さから得ることができる．

段階 4　M_{12} の模倣がまだ完了していないときには，この段階 4 において M_{12} の残りのステップを M_{12} の動作そのままに模倣する．M_{12} が停止したら，M_1 は無駄な動作を $r-3$ ステップ追加実行して停止する．段階 4 に使う時間は (M_{12} の残りのステップ数) $+\,r-3$，すなわち，

$$f_1(n)+f_2(n)-r\lceil f_2(n)/(r-3)\rceil - r(\lceil n/(r-3)\rceil+1)+r-3$$

である．$f_1(n) \geq cf_2(n)+(1+c)n$ であるから，r が十分大きいときには，この値は正である．段階 1 ～ 段階 4 それぞれにかかる時間をすべて加えるとちょうど $f_1(n)$ である．よって，$f_1(n)$ は時間構成可能である．　□

関数 $f(n)$ が $g(n)$**-時間構成可能**であるとは，1^n を入力したとき $1^{f(n)}$ を $g(n)$ 時間以内で出力する DTM が存在することをいう．同様に，$f(n)$ が $g(n)$**-領域構成可能**であることが定義される．

定理 3.7 (小林の定理) * $f(n)$ は $f(n) \geq (1+c)n$ を満たす実数 $c > 0$ が存在するような関数とする．このとき，$f(n)$ が時間構成可能関数である必要十分条件は $f(n)$ が $O(f(n))$-時間構成可能なことである．

証明 $f(n)$ が DTM M によって時間構成可能であるならば，M の動作をそっくり模倣するのと並行して 1 ステップごとに 1 を出力テープに書き出す DTM M' を構成することができる．M' は $1^{f(n)}$ をちょうど $f(n)$ 時間で出力する．

逆に，1^n を入力としたとき $1^{f(n)}$ を $O(f(n))$ 時間で出力する DTM を M としよう．M の計算時間を $g(n)$ とする．$g(n) \leq df(n)$ を満たす定数 $d > 0$ が存在する．$f_1(n) = f(n)$，$f_2(n) = g(n)$ とおくと，これらが補題 3.3 を満たすことを示す．定義より，$g(n)$ は時間構成可能関数である．$f(n) + g(n)$ は次のような DTM M' により時間構成可能である．M' は，まず M を模倣しながらステップ数を適当なテープ上に 1 進数で書き出す ($g(n)$ 時間使う)．M が停止したら，書き出した 1 すべてを消去すれば，ちょうど $f(n)$ 時間使う．よって，M' は合計で $f(n) + g(n)$ 時間使う．

最後に，$f(n) \geq cg(n) + (1+c)n$ を満たす実数 $c > 0$ が存在することを示せばよい．c_1, c_2, c_3, c_4 を

$$\begin{aligned} &f(n) \geq (1+c_1)n, && (1+c_1)(1-c_2) > 1, \\ &c_3 = (1+c_1)(1-c_2) - 1, && c_4 = \min\{c_2/d,\ c_3\} \end{aligned}$$

を満たす正の実数とすると，

$$\begin{aligned} f(n) &= c_2 f(n) + (1-c_2)f(n) \\ &\geq (c_2/d)g(n) + (1-c_2)(1+c_1)n \\ &= (c_2/d)g(n) + (1+c_3)n \\ &\geq c_4 g(n) + (1+c_4)n \end{aligned}$$

である． □

定理 3.8 関数 $f(n)$ が領域構成可能である必要十分条件は $f(n)$ が $O(f(n))$-領域構成可能なことである．

証明 定理 3.7 と同様である．読者への演習問題とする． □

問 3.3 (1) n^k, 2^n, $n!$, 2^{n^k} がそれぞれ時間構成可能であることを示せ (k は正整数)．
(2) $f(n)$ が時間構成可能関数なら $\log f(n)$ は $f(n)$-時間構成可能であることを示せ．
(3) 例 3.2 の (3) の残りを証明せよ．また，同様なことが時間構成可能関数についても成り立つことを示せ．

---

*K.Kobayashi, On proving time constructibility of functions, *Theor. Comput. Sci.* 35, pp.215-225, 1985.

3.2.4 単テープ DTM と通過列

与えられた言語を受理する TM を構成するときには TM の能力が高いほど簡単であるが，TM の能力の限界について議論するときには TM はなるべく簡単な方がよい．この節では，1 本のテープしか持たない**単テープ TM** (single-tape TM) について考え，系 3.1 がこれ以上改良できないことを示す．このために有用な手法として**通過列論法**と呼ばれるものがある†.

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を単テープ DTM とし，入力 $x = a_1 a_2 \cdots a_n$ に対する M の計算を考える．M は時刻 $t_1, t_2, \ldots, t_\ell$ に入力テープヘッドがます目 i と $i+1$ の境界を通過するものとし，通過直前の状態がそれぞれ $q_1, q_2, \ldots, q_\ell$ のとき (図 3.6 参照)，$q_1 q_2 \cdots q_\ell$ を x に対する境界 $(i, i+1)$ における**通過列** (crossing sequence) という．境界の通過は，奇数回目は左から右へ，偶数回目は右から左へである．

補題 3.4 語 $x = x_1 \cdots x_n$, $y = y_1 \cdots y_m$ は単テープ DTM M によって受理され，x に対する境界 $(i, i+1)$ における通過列と y に対する境界 $(j, j+1)$ における通過列が一致するとする．このとき，$x_1 \cdots x_i y_{j+1} \cdots y_m$ も M に受理される．

証明 x に対する計算を境界 $(i, i+1)$ で切り (図 3.6)，y に対する受理計算を境界 $(j, j+1)$ で切ったものとつなぎ合わせると，入力 $x_1 \cdots x_i y_{j+1} \cdots y_m$ に対する受理計算になることは明らかであろう (図 3.7). □

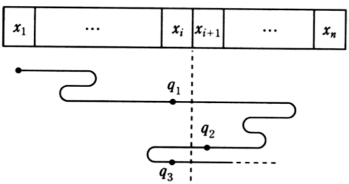


図 3.6 境界 $(i, i+1)$ における通過列

定理 3.9 M が WW^R を受理する $t(n)$ 時間限定単テープ DTM ならば $t(n) \neq o(n^2)$ である．

† F.C.Hennie, One-tape off-line Turing machine computations, *Inform. Contr.* 8, pp.553-578, 1965.

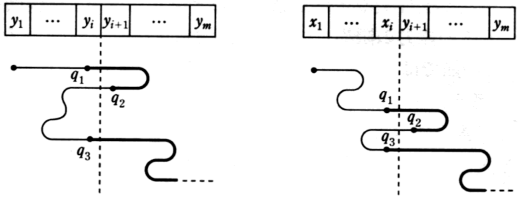


図 3.7 通過列が一致する境界における計算のつなぎ合わせ

証明 長さ $2n$ の入力 ww^R ($w \in \{a,b\}^*$, $|w| = n$) に対する境界 $(i, i+1)$, $1 \leq i \leq n$ における通過列の長さを $\ell_{w,i}$ とし，その平均を a_i とする:

$$a_i = \left(\sum_{w \in \{a,b\}^n} \ell_{w,i} \right) / 2^n.$$

2^n は $|w| = n$ なる $w \in \{a,b\}^*$ の個数である．このとき，少なくとも半分 (すなわち，2^{n-1} 個以上) の w に対して $\ell_{w,i} \leq 2a_i$ でなければならない (そうでないと，$\ell_{w,i}$ の平均は a_i より大きくなってしまう)．M の状態の個数を s とすると，長さが $2a_i$ 以下の通過列の個数は $\sum_{j=0}^{2a_i} s^j \leq s^{2a_i+1}$ であるから，少なくても $2^{n-1}/s^{2a_i+1}$ 個の w が境界 $(i, i+1)$ において同じ通過列を持つ．

一方，$w = w_1 \cdots w_n$ ($w_i \in \{a,b\}$) とするとき，$w_1 \sim w_i$ が同じであるような語は全部で 2^{n-i} 個しかないから，もし $2^{n-1}/s^{2a_i+1} > 2^{n-i}$ とすると，次の条件を満たす語 $ww^R, w'(w')^R$ が存在する:

1) ww^R と $w'(w')^R$ の境界 $(i, i+1)$ における通過列は同じ．

2) ww^R と $w'(w')^R$ は先頭から i 文字以内が異なる．

補題 3.4 より，$w'w^R$ も $w(w')^R$ も M に受理される．これは $w'w^R, w(w')^R \in \mathrm{WW}^R$ を意味し，矛盾である．ゆえに，

$$2^{n-1}/s^{2a_i+1} \leq 2^{n-i} \text{ すなわち } s^{2a_i+1} \geq 2^{i-1}$$

である．よって，$a_i \geq bi - c$ となる定数 $b, c > 0$ が存在する．

M の計算時間は各境界における通過列の長さの和に等しいから，入力 ww^R ($w \in \{a,b\}^*$, $|w| = n$) に対する M の時間量の平均は $\Theta\left(\sum_{i=1}^n a_i\right)$ である．

$$\Theta\left(\sum_{i=1}^n a_i\right) = \Theta\left(\sum_{i=1}^n (bi - c)\right) = \Theta(n^2)$$

である．よって，ある定数 $d > 0$ が存在して，無限に多くの n に対して $t(n) \geq dn^2$ となるので，$t(n) = o(n^2)$ ではない． □

WW^R は $O(n)$ 時間限定 2 テープ DTM で受理される (問 3.4(1)) から，定理 3.9 は，系 3.1 がこれ以上改良できないことを示している．

定理 3.9 と同様に，1 作業テープオフライン DTM の領域量に関する次の定理も通過列論法によって証明することができる (演習問題 2) ‡.

定理 3.10 M が WW^R を受理する $s(n)$ 領域限定の 1 作業テープオフライン DTM ならば $s(n) \neq o(\log n)$ である．

問 3.4 (1) WW^R を受理する n 時間限定の 2 テープ DTM を示せ．
(2) WW^R を受理する $\log_2 n$ 領域限定の 1 作業テープオフライン DTM を示せ．
(3) この節の論法が (少し修正すれば) NTM や次の言語にあてはまるかどうか検討せよ:　(a) WcW　(b) AnBn

3.3　その他の計算量

ヘッド反転量

典型的な計算量としてこれまで時間量や領域量について考察してきた．しかし、計算の複雑さを表わす量が時間量や領域量だけでないことはもちろんである．TM を計算モデルとする場合，そのほかにテープヘッドの反転回数などが計算量として考えられる．簡単のために，単テープ DTM を例にして述べる．

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を単テープ DTM とする．M の時刻 t における遷移が $\delta(q_t, a_t) = (p_t, b_t, d_t)$ であったとする．$d_{t-k}d_t < 0, d_{t-k+1} = \ldots = d_{t-1} = 0$ $(k \geq 1)$ であるとき，M のテープヘッドは時刻 t に**反転**したという．長さ n の任意の入力 に対して M の計算におけるテープヘッドの反転回数が $r(n)$ 以下であるとき，M のヘッド**反転量** (reversal complexity) は $r(n)$ であるという．NTM の場合には，M に受理される任意の入力 x に対してテープヘッド反転回数が $r(|x|)$ 以下であるような受理計算が存在するなら M は $r(n)$ **ヘッド反転限定** ($r(n)$ reversal bounded) であるという．また，オフライン TM の場合には入力テープ上のヘッド反転回数はカウントしないものとする．

例 3.3　　例 2.4 のオフライン DTM M のヘッド反転量は $4n$ である (同時に M の領域量は $O(\log n)$ である)．$O(n)$ 領域が使えるなら，同じ言語 AnBn を受理する DTM でヘッド反転量が 1 のものが存在する (問 2.4 (3) への補遺: 証明せよ)．　□

例えば，次のことが知られている§．

---

‡A.Cobham, Time and memory requirements for machines which recognize squares or palindromes, *IBM Res. Rept.* RC-1621, 1966.

§この節の命題の証明はすべて省略する．出典を挙げておくので参照されたい．

S.A.Greibach, Visits, crosses, and reversals for nondeterministic off-line machines, *Inform. Contr.* 36, pp.174-216, 1978.

命題 3.1 $f(n)$ ヘッド反転限定の 1 作業テープオフライン NTM が受理する言語のクラスを $\mathbf{NREVERSAL}(f(n))$ で表わす.

(1) $\mathbf{NREVERSAL}(f(n)) = \mathbf{NSPACE}(f(n)\log n)$.

(2) $f(n)$ は領域構成可能関数で $f(n+1) = O(f(n))$ を満たすとする。このとき，$\mathbf{NREVERSAL}(f(n)) \subsetneq \mathbf{NREVERSAL}(kf(n))$ となる整数 $k > 1$ が存在する．すなわち，ヘッド反転量は線形縮小できない．

同時計算量

これまでは計算量を考えるとき，時間量だけ，領域量だけ，あるいはヘッド反転量だけに注目した．しかし，これらは独立のものではなく，時間量を小さくしようとすると領域量が大きくなってしまうとか，逆に，領域量を小さくしようとすると時間量が大きくなってしまうとかいうように，**トレードオフ** (trade off) がある．読者は，例えば WW^R を受理する 1 作業テープオフライン DTM の時間量と領域量をできるだけ小さくすることを考えてみるとよい．問 3.4 に述べたように，時間量 $t(n)$ は $t(n) = O(n)$ とできるし，領域量 $s(n)$ も $s(n) = O(\log n)$ とできる．しかし，これらを同時に満たすことはできない (実は，$t(n)s(n) \neq o(n^2)$ が成り立つ: 演習問題 6).

いくつかの計算量を同時に考えたものを**同時計算量** (simultaneous complexity) という．$t(n), s(n), r(n)$ をそれぞれ時間量，領域量，ヘッド反転量とするとき，次のようなクラスが考えられる:

$$\mathbf{DTISP}(t(n), s(n)) = \{L \mid L \text{ を受理する } t(n) \text{ 時間限定かつ } s(n) \text{ 領域限定の DTM が存在する}\}.$$

同様に，$\mathbf{DRESP}(r(n), s(n))$, $\mathbf{DTIRE}(t(n), r(n))$ が定義される．また，NTM に関して $\mathbf{NTISP}(t(n), s(n))$ 等も同様に定義される．各計算量の間のトレードオフの例として次の命題を述べておこう¶.

命題 3.2 1 作業テープオフライン NTM に対し，次のことが成り立つ:

$$\begin{aligned}\mathbf{NTISP}(t(n), s(n)) &\subseteq \mathbf{NRESP}(s(n), t(n)) \\ &\subseteq \mathbf{NTISP}(s(n)t(n)n^2\log n, s(n)\log n).\end{aligned}$$

公理論的計算量

M.Blum は，計算量とは何かを次のように公理化し，いろんな計算量に共通する性質を統一的・一般的に導いた‖.

¶E.Moriya, S.Iwata and T.Kasai, A note on some simultaneous relations among time, space, and reversal for single work tape nondeterministic Turing machines, *Inform. Contr.* 70, pp.179-185, 1986.

‖M.Blum, A machine-independent theory of the complexity of recursive functions, *J. Assoc. Comput. Mach.* 14, pp.322-336, 1967.

すべての TM を列挙したものを $M_1, M_2, \ldots$ とする (i が与えられたとき M_i が構成的に求められることが必要である．いろんな列挙の仕方が可能であるが，例えば，例 2.5 (4) のように 2 進数で符号化し，2 進数としての値が小さい順に列挙すると，この条件を満たす列挙となる)．2.4 節で述べたように，各 M_i を $\mathbf{N}$ から $\mathbf{N}$ への部分関数 ϕ_i とみなす (命題 2.2 により，$\phi_1, \phi_2, \ldots$ は 1 変数の帰納的部分関数全体の列挙である (重複あり))．M_i は ϕ_i を計算するアルゴリズムである．$\Phi = \langle \Phi_1, \Phi_2, \ldots \rangle$ を 1 変数の帰納的部分関数の列とする．次の 2 つの公理を満たすとき，Φ のことを**計算量の測度** (complexity measure)，とくに Φ_i のことを ϕ_i に付随する計算量という．

公理 1 $\Phi_i(x)$ が定義されるのは，$\phi_i(x)$ が定義されるとき，かつそのときだけである．

公理 2 i, x, y が与えられたとき，$\Phi_i(x) = y$ なら $R(i, x, y) = 1$，$\Phi_i(x) \neq y$ なら $R(i, x, y) = 0$ となる帰納的関数 R が存在する．

公理 1 は，i 番目のアルゴリズム (TM) ϕ_i に入力 x を与えたときに使用される資源 (時間や領域など) の量 $\Phi_i(x)$ が定義されるのは，そのアルゴリズムが入力 x に対して停止するとき，かつそのときに限ることを主張している．

公理 2 は，アルゴリズム ϕ_i に入力 x を与えたとき，使用した資源の量がちょうど y であるかどうかを判定する停止性のアルゴリズム (R のこと) が存在することを主張している．

例 3.4 (1) DTM の時間量 $\Phi_i(x) =$「TM Mに入力 x を与えたときの停止までのステップ数」が Blum の公理を満たす測度であることは明らかである．R は，入力 i, x, y (のコード) が与えられたら，TM M_i を入力 x に対して模倣しながら 1 ステップごとに y から 1 減じ，0 になったときにちょうど M_i が停止したら 1 を出力し，そうでなかったら 0 を出力して停止するような TM により計算可能である．

(2) DTM の領域量は Blum の公理 1 を満たさない．なぜなら，使用する領域量は有限であっても計算がループに陥ることがあるからである．しかし，これは本質的なことではなく，計算が停止しないときは領域量は定義されないことにすればよい (時間量や領域量を定義するとき，停止性の TM しか考えないことにしてもよい)．　□

問 3.5 次の計算量はいずれも Blum の公理を満たす測度であることを示せ．
(a) NTM の時間量と領域量　(b) DTN (NTM) のヘッド反転量

すでに線形加速定理や線形領域圧縮定理で証明したように，どんな TM の時間量も領域量も何分の 1 にでも小さくできる．これはたかだか定数分の 1 の話

であるが，ある特定の言語に対しては時間量や領域量を任意に小さくできる (すなわち，時間量や領域量が最小である TM が存在しない) ことが Blum の公理から証明できる．

命題 3.3 (加速定理) Blum の公理を満たす任意の測度 Φ および任意の帰納的関数 r に対して，次の性質を満たす帰納的関数 f が存在する: f を計算するどんなアルゴリズム ϕ_i も r 倍加速できる．すなわち，f を計算する別のアルゴリズム ϕ_j が存在して，$r(\Phi_j(n)) \leq \Phi_i(n)$ がほとんどすべての n に対して成り立つ．

「ほとんどすべての」とは，「有限個の例外を除いて」という意味である．TM を言語の受理機と考え，Φ を例えば時間量とみたとき，この定理は次のように述べ直すことができる: 任意の帰納的関数 r に対して，帰納的言語 L が存在し，L を受理する任意の TM M に対し L を受理する別の TM M' が存在して，
r(長さ n の入力に対する M' の時間量) $\leq$ (長さ n の入力に対する M の時間量)
がほとんどすべての n に対して成り立つ．

命題 3.3 より，f を計算する最良のアルゴリズム (L を受理する TM で時間量が最小のもの) は存在しない，という奇妙な結果が導かれる．しかし，そのような f や L は実は特殊なものであって，たいていの f や L に対しては最良のアルゴリズムが存在する．g を 2 変数の帰納的関数とし，Φ を Blum の公理を満たす測度とする．1 変数関数 f を計算するアルゴリズム ϕ_i で $\Phi_i(n) \leq g(n, f(n))$ を満たすものが存在するとき，f は g **に従順** (g-honest) であるという．とくに，$g(n, f(n)) = cf(n)$ (c は定数) であるとき $f(n)$ は**線形従順** (linearly honest) であるという．Φ が時間量 (領域量) のとき，線形従順な関数とは時間 (領域) 構成可能関数のことに他ならない．f が帰納的関数 g に従順であるならば，f を任意に加速することはできないことが知られている．

次の定理は 4 章で述べる，時間量や領域量に関する階層定理がもっと一般に，Blum の公理を満たす任意の測度に対して成り立つことを示している．

Φ を Blum の公理を満たす測度，$r(n)$ を帰納的関数とするとき，

$$\mathcal{C}^{\Phi}_{r(n)} = \{f \mid f \text{ を計算するアルゴリズム } \phi_i \text{ が存在して，} \\ \text{ほとんどすべての } n \text{ に対して } \Phi_i(n) \leq r(n)\}$$

と定義する．例えば，Φ が DTM の時間量なら $\mathcal{C}^{\Phi}_{r(n)} = \mathbf{DTIME}(r(n))$ である．

命題 3.4 Blum の公理を満たす任意の測度 Φ および任意の帰納的関数 $g(n)$ に対して，次のような帰納的関数 $h(n)$ が存在する: $t(n)$ が g に従順な関数なら，$\mathcal{C}^{\Phi}_{t(n)}$ の元ではないが $\mathcal{C}^{\Phi}_{h(t(n))}$ の元であるような関数が存在する．

例えば，Φ を TM の領域量とし，$g(n)$ を領域構成可能関数とするとき，$\mathbf{DSPACE}(t(n)) \subsetneq \mathbf{DSPACE}(h(t(n)))$ となる $h(n)$ を求めることができる．

一方，次の間隙定理 (Gap Theorem) は，計算量のクラスの間には任意に大きい間隙が存在しうることを言っている**．

命題 3.5 (間隙定理) Blum の公理を満たす任意の測度 Φ および任意の帰納的関数 $g(n) \geq n$ に対して，$\mathcal{C}^{\Phi}_{t(n)} = \mathcal{C}^{\Phi}_{g(t(n))}$ となる帰納的関数 $t(n)$ が存在する．

最後に，次の和定理 (Union Theorem) は，いくつかの計算量のクラスの和集合が再び計算量のクラスになることを保証するものである‡．例えば，5 章で導入するクラス $\mathbf{P} = \bigcup_{k\geq 0} \mathbf{DTIME}(n^k)$ に対しても $\mathbf{P} = \mathbf{DTIME}(t(n))$ となる帰納的関数 $t(n)$ が存在する．

命題 3.6 (和定理) $f_1, f_2, \ldots$ は $f_i(n) < f_{i+1}(n)$ が任意の i, n に対して成り立つような帰納的関数とする．Blum の公理を満たす任意の測度 Φ に対して，$\bigcup_{i\geq 1} \mathcal{C}^{\Phi}_{f_i(n)} = \mathcal{C}^{\Phi}_{t(n)}$ となる帰納的関数 $t(n)$ が存在する．

演 習 問 題 3

1 (1) 言語 L が NTM で生成されるならば L を生成する DTM が存在することを示せ．

(2) L が句構造文法で生成されるならば L を生成する NTM が存在することを示せ．逆に，L が NTM で生成されるならば L を生成する句構造文法が存在することも示せ．(1), (2) より，命題 2.3 が証明される．

2 $\mathrm{WW}^R \notin \mathbf{DSPACE}(o(\log n))$ であることを示せ (ヒント: 定理 3.9 と同様)．$\mathbf{DSPACE}(o(f(n)))$ は $\bigcup_{g(n)=o(f(n))} \mathbf{DSPACE}(g(n))$ を表わす．

3 NTM の時間量の定義において，長さ n の任意の入力 w に対して，w に対する任意の計算の長さが $t(|w|)$ 以下であるとき，M は $t(n)$ **時間限定**であると定義しなおす．同様に，領域量の定義も変更する．このような定義の変更が，この章の定理にどのように影響するか考察せよ (2 章の演習問題 5 参照)．とくに，問 3.2 および定理 3.1, 3.6 について考察せよ．

3.1 節で述べているような定義以外に，文献では本問のような定義もよく用いられている．2 章の演習問題 5 の場合も含めて，両定義の間に計算量の本質を変えるような差

** A.Borodin, Computational complexity and the existence of complexity gaps, *J. Assoc. Comput. Mach.* 19, pp.158-174, 1972.

‡ E.M.McCreight and A.R.Meyer, Classes of computable functions defined by bounds on computation, *Proc. 1st ACM Symp. on Theory of Comput.*, pp.79-88, 1969.

はないが，同じというわけではない．$t(n)$ や $s(n)$ が時間 (領域) 構成可能な場合には，定義される計算量のクラスが一致するという意味で両定義は等しい．

4 3.2.3 節で定義した "構成可能性" は，文献によっては "fully time (space) constructible" と呼ばれるものである．その場合，もう少し弱い条件を "time (space) constructible" の定義とする．すなわち，任意の $n \in \mathbf{N}$ に対して長さ n の入力 w が存在し，w に対する計算がちょうど $f(n)$ ステップで停止するような DTM M が存在するとき，$f(n)$ は time constructible であるという．space constructibility も同様に定義される．$f(n) \geq n$ であるとき，$f(n)$ が space constructible なら fully space constructible であることを示せ．

5 単テープ TM の線形加速 (定理 3.4) はできるか？

6 (通過列論法の拡張) $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を k 作業テープオフライン TM とする．入力テープヘッドがます目 i と $i+1$ の境界を通過したときの状態が q, 作業テープの内容とヘッド位置が $(\alpha_1, h_1, \ldots, \alpha_k, h_k)$ のとき，$(p, \alpha_1, h_1, \ldots, \alpha_k, h_k)$ を**全状態**と呼ぶ．通過列として，状態 q だけの代わりに全状態を考えることにより，定理 3.9 を次のように拡張せよ:

定理 M を WW^R を受理する $t(n)$ 時間限定かつ $s(n)$ 領域限定の k 作業テープオフライン TM とすると，$t(n)s(n) \neq o(n^2)$ である．

7 $O(1)$ 領域限定の単テープ TM (入力テープしかないようなオフライン TM に等しい) のことを **2 方向有限オートマトン** (two-way finite automaton) という．とくに，入力テープヘッドが右方向にしか動かないような 2 方向有限オートマトンを単に有限オートマトン (**FA**) と呼ぶ．言語 L が 2 方向 FA で受理されるなら，L を受理する FA が存在することを通過列論法を用いて証明せよ．

ちなみに，FA が受理する言語と正則集合 (2.5 節参照) とは一致することが知られている．また，決定性 FA と非決定性 FA とは受理能力が等しいことが知られている*.

8 L が $o(\log\log n)$ 領域限定 NTM M で受理されるなら L は FA で受理されることを示せ．ヒント: M 本来の状態と作業テープの内容を一緒にしたものを "全状態" と呼ぶ (問題 6 参照)．M は全状態を状態とする 2 方向 FA と考えることができる．領域を s だけ必要とする，長さが最小の入力 w を考える．w のプレフィックスはすべて異なる動作 (全状態の遷移) を引き起こす (さもないと，w の最小性に反す)．ところが，このような動作の仕方はたかだか $c_1^{c_2^s}$ 通りしかない ($c_1, c_2 \geq 2$ は定数)．

9 Blum の公理を満たす測度となるか？

(a) $\Phi_i(n) =$ 長さ n の入力に対して M_i が状態を変えた回数

(b) $\Phi_i(n) =$ 長さ n の入力に対して M_i が遷移にあたって，読んだ記号と異なるテープ記号をその上に書き込んだ回数

10 3.3 節で時間量と領域量の間のトレードオフの例を見た．このようなことは Blum の公理を満たす任意の測度 Φ と Φ' の間で成り立つ．任意のアルゴリズム ϕ_i と任意の $n \geq i$ に対して $\Phi(n) \leq h(n, \Phi'_i(n))$ かつ $\Phi'_i(n) \leq h(n, \Phi_i(n))$ となる計算可能関数 $h(n)$ が存在することを証明せよ．

11 1 テープ TM M と M への入力 $w \in \{0,1\}^*$ を例 2.5 (4) のように符号化した

*M.O.Rabin and D.Scott, Finite automata and their decision problems, *IBM J. Res. Develop.* 3, pp.115-125, 1959.

ものを $\langle M, w\rangle$ で表わす．$x \in \{0,1\}^*$ とする．M に w を入力として与えるとテープ上に x を残して停止するならば，$\langle M, w\rangle$ を x の**記述** (description) という．

$$K(x) = \min\{|\langle M, w\rangle| \mid \langle M, w\rangle \text{ は } x \text{ の記述}\}$$

を x の最小記述量 (descriptive complexity) とか，**コルモゴロフ計算量** (Kolmogorov complexity) という[†]．

ある定数 $c > 0$ が存在して，任意の x に対して $K(x) \leq |x| + c$ が成り立つ．なぜなら，任意の入力 x に対して何もせずにただちに停止する TM を M とすると，$\langle M, x\rangle$ が x の記述となるからである．

ある事柄 x を相手に伝えるとき，もし x になんらかの規則性がある場合には，その規則を説明すれば x が何かを簡潔に伝えることができる．それに対し，x に何の規則性もない場合には，x 自身をそのまま伝えるしかないであろう．例えば，次の x_1 と x_2 の記述 $\langle M, w\rangle$ を考えてみるとよい:

$$x_1 = 010101010101010101010101010101010101010101010101010101010101010101010101$$
$$x_2 = 0100110101010001101111111111001010101010111010101010001010101111010100000$$

この意味で，$K(x)$ が $|x|$ に近いほど x はランダムなものと考えることができる．

(1) 次のそれぞれが任意の x, y に対して成り立つような正整数 c_1, c_2, c_3 が存在することを示せ．

(a) $K(xx) \leq K(x) + c_1$　　(b) $K(xy) \leq 2K(x) + K(y) + c_2$

(c) $K(xy) \leq 2\log_2 K(x) + K(y) + c_3$

(2) $K(x) \leq |x| - c$ となる正整数 c が存在するとき，x は **c-圧縮可能** (c-compressible) であるという．c-圧縮可能ではない語が少なくとも $2^n - 2^{n-c+1} + 1$ 個存在することを示せ．

(3) $K(x)$ は計算可能関数ではないことを示せ．

[†] A.N.Kolmogorov, Three approaches for defining the concept of information quantity, *Probl. Information Transmission* 1, pp.1-7, 1965.　同じことを独立に Chaitin も発見したので，Kolmogorov complexity のことを Kolmogorov–Chaitin complexity ともいう:

G.J.Chaitin, On the length of programs for computing finite binary sequences, *J. Assoc. Comput. Mach.* 13, pp.547-569, 1966.

4

計算量のクラスの階層

この章では計算量のクラスの間の一般的関係について述べる．とくに，時間量や領域量のクラスに無限の階層が存在することを示す．このことはすでに 3 章の命題 3.4 で，TM の時間量や領域量に限らずもっと一般の計算量に対して成り立つことを述べたが，この章ではそれを具体的に TM の時間量，領域量について考察する．証明には対角線論法と呼ばれる，計算量の理論で典型的な手法の一つを用いる．また，この章の後半では決定不能な問題について考察する．決定不能問題とは，それを解くためのアルゴリズム (すなわち，それを符号化した言語を受理する停止性 TM) が存在しないような決定問題のことである．したがって，決定不能問題 (を符号化した言語) は帰納的でなく，どの計算量のクラスにも属さない．

4.1 計算量のクラスの間の基本的関係

まず，前章で導入した **DTIME**, **DSPACE**, **NTIME**, **NSPACE** に関していくつかの興味深いクラスを定義しよう．

$\mathbf{REG}{=}\mathbf{DSPACE}(1)$

$\mathbf{DL} = \mathbf{DSPACE}(\log n)$, $\quad \mathbf{NL} = \mathbf{NSPACE}(\log n)$

$\mathbf{P} = \bigcup_{i\geq 1} \mathbf{DTIME}(n^i)$, $\quad \mathbf{NP} = \bigcup_{i\geq 1} \mathbf{NTIME}(n^i)$

$\mathbf{PSPACE} = \bigcup_{i\geq 1} \mathbf{DSPACE}(n^i)$,

$\mathbf{EXPSPACE} = \bigcup_{i\geq 1} \mathbf{DSPACE}(2^{n^i})$

$\mathbf{DEXP} = \bigcup_{i\geq 1} \mathbf{DTIME}(2^{n^i})$, $\quad \mathbf{NEXP} = \bigcup_{i\geq 1} \mathbf{NTIME}(2^{n^i})$

$\mathbf{REC} = \{L(M) \mid M$は停止性 DTM$\}$

$\mathbf{RE} = \{L(M) \mid M$は任意の DTM$\}$

また，$\mathcal{C}$ を言語のクラスとするとき

$$\text{co-}\mathcal{C} = \{L^c \mid L \in \mathcal{C}\}$$

と定義する．ここに，L^c は L の補集合を表わす．

REG は正則集合のクラスに等しく，**DSPACE**(1) = **NSPACE**(1) であることが知られている (3 章の演習問題 7 参照)．**DL**, **NL** は文献によってはそれぞれ **DLOG**, **NLOG** と表わされている．**REC** は帰納的言語のクラスのことであり，**RE** は帰納的可算言語のクラスのことである．任意の $f(n)$ に対して，

$$\mathbf{DTIME}(f(n)) \subseteq \mathbf{NTIME}(f(n)), \quad \mathbf{DSPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$$
$$\mathbf{DTIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n)), \mathbf{NTIME}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$$

が成り立つことは定義から自明である．なお，本書では証明しないが，$f(n)$ が時間構成可能関数であるなら $\mathbf{DTIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n)/\log f(n))$ であることが知られている*.

定理 4.1 (1) 任意の $f(n)$ に対して $\mathbf{NTIME}(f(n)) \subseteq \bigcup_{c>0} \mathbf{DSPACE}(c^{f(n)})$. とくに，$f(n)$ が時間構成可能または領域構成可能な関数なら $\mathbf{NTIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n))$ である．

(2) $\mathbf{NSPACE}(f(n)) \subseteq \bigcup_{c>0}\mathbf{DTIME}(c^{\log n + f(n)})$ が成り立つ．したがって，$\mathbf{DSPACE}(f(n)) \subseteq \bigcup_{c>0}\mathbf{DTIME}(c^{\log n+f(n)})$ も成り立つ．

証明 (1) 定理 3.1 より，$\mathbf{NTIME}(f(n)) \subseteq \bigcup_{c>0}\mathbf{DTIME}(c^{f(n)})$. したがって，上述の包含関係より $\mathbf{NTIME}(f(n)) \subseteq \bigcup_{c>0}\mathbf{DSPACE}(c^{f(n)})$.

$f(n)$ が時間構成可能である場合には，$f(n)$ 時間限定 NTM M を模倣する DTM M' を次のように作る: 長さ n の入力に対し，M' は最初に各テープ上の大きさ $f(n)$ の領域に印を付ける ($f(n)$ が時間 (領域) 構成可能だから，そのようにできる)．次いで，問 3.2 に述べたのと同じ方法で模倣を行なう．すなわち，長さが $f(n)$ 以内の M の動作命令の列を系統的に生成し，それが M の受理計算になるかどうかを検証する，ということを繰り返す．1 つでもそのような受理計算が見つかったら直ちに停止して入力を受理するが，生成した動作命令の列が M の受理計算でなかった場合は，すでに使った領域を消去して，そこに次の動作命令の列を生成して模倣をつづける．$O(f(n))$ 領域を使えば，長さ $f(n)$ 以内の M の動作命令の列すべてを系統的に順次生成できる．

(2) 定理 3.2 より，M は 1 作業テープオフライン NTM と仮定してもよい．M の状態の個数を s，テープ記号の個数を t とすると，長さ n の入力 x に対して M が取り得る ID の個数はたかだか $(n+2)\cdot s\cdot f(n)\cdot t^{f(n)}$ である．$(n+2)\cdot s\cdot f(n)\cdot t^{f(n)} \le d^{\log n+f(n)}$ となる正整数 d が存在する．

---

*J.E.Hopcroft, W.J.Paul and L.G.Valiant, On time vs. space and related problems, *Proc. 16th Annual IEEE Symp. on the Foundtn. Comput. Sci.*, pp.57-64, 1975.

入力 x に対する M の ID を頂点とし，$(I_1, I_2) \in E(G_{M,x}) \Longleftrightarrow I_1 \vdash_M I_2$ と定義された有向グラフ $G_{M,x}$ を入力 x に対する M の**様相グラフ** (configuration graph) と呼ぶ．M と x から $G_{M,x}$ のコード $\overline{G_{M,x}}$ を次のように生成する DTM M_1 を構成することができる．$\overline{G_{M,x}}$ は辺すなわち隣接する 2 頂点の対 (I_1, I_2) すべてを列挙した文字列

$$\overline{G_{M,x}} = \cdots \#\# I_1 \# I_2 \#\# \cdots$$

で表わす．I_1, I_2 は M の ID (を適当に符号化した文字列) であり，$\#$ は区切りを表わすための特別な記号である．M の ID 1 つの長さは $O(f(n))$ であり，$G_{M,x}$ の頂点 ($= M$ の ID) はたかだか $d^{\log n + f(n)}$ 個しかないから $|\overline{G_{M,x}}| = O((d^{\log n + f(n)})^2 f(n))$ とできる．M_1 は定理 3.1 の証明と同様な方法で $G_{M,x}$ の辺すなわち (I_1, I_2) すべてを重複しないようにチェックしながら生成する (ついでに，次のステップで $G_{M,x}$ における到達可能性問題を解くときに使うために，全頂点のリストを別のテープ上に生成しておく)．定理 3.1 の証明と同じ論法で，M_1 の時間量は $O(|\overline{G_{M,x}}|^2)$ であることがいえる．

例 1.10 で見たように，有向グラフの到達可能性問題を解く，時間量が O(頂点数 + 辺数) のアルゴリズムが存在するが，このアルゴリズムの時間量は，有向グラフの任意の頂点から隣接する頂点を求めることや任意の頂点がすでにたどられたか否かを判定することがそれぞれ $O(1)$ 時間でできるとした場合のものである．$G_{M,x}$ は M_1 が生成した文字列 $\overline{G_{M,x}}$ と全頂点のリストを表わす文字列 (その長さは $O(|\overline{G_{M,x}}|)$ である) とで与えられるが，この $G_{M,x}$ に対する到達可能性問題を解く $O(|\overline{G_{M,x}}|^2)$ 時間限定 DTM M_2 を構成することができる．

以上の議論により，x が M に受理されるかどうかを時間量 $O(|\overline{G_{M,x}}|^2) = O(c^{\log n + f(n)})$ で判定する DTM が存在することが示された ($c > 0$ は適当な定数)． □

系 4.1 **DL** $\subseteq$ **NL** $\subseteq$ **P** $\subseteq$ **NP** $\subseteq$ **PSPACE**.

問 4.1 (1) 定理 4.1 の (1) の証明において，$f(n)$ が構成可能のとき，$O(f(n))$ 領域を使って長さ $f(n)$ 以内の M の動作命令の列すべてを系統的に生成する方法を示せ．

(2) 定理 4.1 の (2) の証明において，$d^{\log n + f(n)}$ の $\log n$ はなぜ必要なのか？ また，M_2 をもっと具体的に示せ．

NTM を DTM で模倣する際，時間量に関しては定理 3.1 に述べたような自明な模倣 ($\mathbf{NTIME}(t(n)) \subseteq \bigcup_{c>0} \mathbf{DTIME}(c^{t(n)})$) しか知られていないが，領域量に関してはもっと良い模倣の方法が存在する．

定理 4.2 (Savitch の定理) $s(n) \geq \log n$ を領域構成可能関数とする．M が $s(n)$ 領域限定 NTM なら，M を模倣する $s(n)^2$ 領域限定 DTM M' が存在する．

証明 M の作業テープの本数を k とする．M の状態の個数を q，テープ記号の個数を r とすると，長さ n の入力に対して，M の異なる ID の個数はたかだか $q\cdot(n+2)\cdot s(n)^k\cdot r^{ks(n)}$ である．$s(n)\geq\log n$ であるから，$q\cdot(n+2)\cdot s(n)^k\cdot r^{ks(n)}\leq c^{s(n)}$ が任意の $n\geq 1$ に対して成り立つような定数 c が存在する．長さ n の入力 x が与えられたとき，M' は $O(s(n))$ 領域を使い，これらの ID を系統的に順次生成することができる．

I_1, I_2 を M の ID とする．M が I_1 からたかだか 2^i ステップで I_2 に到達できるならば $I_1 \stackrel{(i)}{\Longrightarrow} I_2$ と書くことにする．定義より，$i\geq 1$ のとき $I_1 \stackrel{(i)}{\Longrightarrow} I_2$ が成り立つ必要十分条件は $I_1 \stackrel{(i-1)}{\Longrightarrow} I_3 \stackrel{(i-1)}{\Longrightarrow} I_2$ となる I_3 が存在することである．このことに基づき，M' は入力 x に対する M の初期 ID (それを $\mathrm{Init}_{M,x}$ とする) から受理 ID へ到達できるかどうかを再帰的に判定する:

```
procedure M'(x)
begin
    h ← ⌈log₂ c⌉s(n);
    forall 長さ s(n) 以下の M の受理 ID I_f do
        if reachable(Init_{M,x}, I_f, h) then x を受理して停止する end
    end;
    x を拒否する
end

function reachable(I_1, I_2, i)
begin /* I_1 =(i)=> I_2 かどうか判定する */
    if i = 0 and (I_1 = I_2 or I_1 ⊢_M I_2) then return true end;
    if i ≥ 1 then
        forall 長さ s(n) 以下の M の ID I_3 do
            if reachable(I_1, I_3, i − 1) and reachable(I_3, I_2, i − 1) then
                return true
            end
        end
    end;
    return false
end
```

reachable(I_1, I_2, i) の計算木の高さ (＝再帰の最大深さ) は i であり，ステップ数が 2^i 以下の M の計算がそれに対応する．入力 x に対する M の ID の個数 $\leq c^{s(n)}$ だから，x が M に受理される必要十分条件は，ステップ数 $\leq c^{s(n)}$ なる受理計算が存在することである．よって，M' は $\mathrm{Init}_{M,x} \stackrel{(h)}{\Longrightarrow} I_f$ かどうかをすべての受理 ID I_f に対して調べれば十分である．ここに，$h=\lceil\log_2 c^{s(n)}\rceil$ である．以上の考察より，$L(M')=L(M)$ である．

最後に M' の領域量を考えよう．関数 reachable(I_1, I_2, i) の変数 (reachable の中だけで局所的に使われている変数) は I_1, I_2, i だけだから，その領域量は $O(2s(n)+\log h) = O(s(n))$ である (i は 2 進数で表わすので，必

要な領域量は $O(\log i)$ である). したがって, 系 1.1 より, M' の領域量は「reachable$(\mathrm{Init}_{M,x}, I_f, h)$ の計算木の高さ」×「reachable の局所変数の領域量」$= O(h \cdot s(n)) = O(s(n)^2)$ である. □

系 4.2 $\mathbf{PSPACE} = \bigcup_{i \geq 1} \mathbf{NSPACE}(n^i)$.

2.4 節で, DTM M を変換機と考え, 入力 x に対する出力 $M(x)$ を定義した. M が NTM の場合には, $M(x)$ は次のように定義される言語である:

$y \in M(x) \Longleftrightarrow M$が x を受理し停止したときの出力テープの内容が y.

DTM の場合と同様に, 出力テープの使用量は領域量にカウントしない.

さて, グラフの到達可能性問題 (有向グラフ G と頂点 s と t が与えられたとき, s から t へ到達可能か否かを判定する問題) は $O(\log n)$ 領域限定の非決定性アルゴリズム (NTM) で解くことができる (問 4.2). この問題を拡張して, x から到達可能な頂点の個数を求めることも $O(\log n)$ 領域限定の非決定性アルゴリズムで解くことができることを以下で証明する. このような, 個数を求める問題を一般に**数え上げ問題** (counting problem) という.

問 4.2 グラフの到達可能性問題を解く $O(\log n)$ 領域限定の非決定性アルゴリズムを示せ. 例 1.10 のアルゴリズムは $O(\log n)$ 領域限定ではないのか?

補題 4.1 グラフ G と頂点 x が与えられたとき, G において x から到達可能な頂点の個数 $M(G,x)$ を出力する $O(\log n)$ 領域限定の NTM M が存在する.

証明 G の頂点数を n とする. x から長さ k 以下の道に沿って到達可能な頂点の集合を $S(k)$ で表わす. 明らかに $|S(0)| = 1$ であり, $|S(n-1)|$ は x から到達可能な頂点の個数である. 以下の非決定性アルゴリズム (NTM M) では $|S(0)|, |S(1)|, \cdots$ を順次計算するが, 各 k について, $|S(k)|$ の計算には $|S(k-1)|$ しか使わない.

はじめに, 頂点 v が与えられたとき, $v \in S(k-1)$ かどうかを $O(\log n)$ 領域で非決定的に判定するアルゴリズムを考える:

```
function member_guess(G, x, v, k)
begin
   1. v_0 ← x;
   2. for i ← 1 to k − 1 do
      2.1. G の頂点を 1 つ非決定的に選び v_i とする;
      2.2. if (v_{i−1}, v_i) ∉ E(G) then 入力を拒否して停止 end
   3. end;
   4. if v_{k−1} = v then return true else 拒否して停止 end
end
```

$v \in S(k-1)$ であるにもかかわらず，member_guess が行 2.1 において v_i の非決定的推測に失敗した結果，入力が拒否されアルゴリズムが停止してしまうことがありうることに注意する．この関数が使う領域量のうち n に依存するものは，i を記憶しておくための $O(\log n)$ だけである．

次に，頂点 u が与えられたとき $u \in S(k)$ かどうかを $|S(k-1)|$ だけを使って判定する非決定性アルゴリズムを示す (関数 path_exist では，$|S(k-1)|$ をあたかも変数のごとく扱っているが，これはあくまでも分かりやすさを重視したための便法である)．

```
function path_exist(G, x, u, k, |S(k − 1)|)
begin
  1. p ← 0; /* S(k − 1) の元のうち，考慮済みのものの個数 */
  2. path_found ← false; /* x から u への道があったか？ */
  3. forall v ∈ V(G) do
     3.1. if member_guess(G, x, v, k) then /* v ∈ S(k − 1) のとき */
          3.1.1. p ← p + 1;
          3.1.2. if (v, u) ∈ E(G) then path_found ← true end
     3.2. end
  4. end;
  5. if p < |S(k − 1)| then 拒否して停止 else return path_found end
end
```

$|S(k-1)|$ が正しく求められているとしよう．関数 path_exist は，行 3.1.1 において $v \in S(k-1)$ であるような v の個数をカウントし，行 3.1.2 において v と u が隣接していたら (このとき $u \in S(k)$ である) 変数 path_found を **true** にする．行 3 の forall 文が終了したとき $p = |S(k-1)|$ であれば，$S(k-1)$ のすべての頂点が v として考慮されたことを意味し，そのうちの少なくとも 1 つの v と u とが隣接していたら (この場合，$u \in S(k)$ である) 行 3.1.2 によって変数 path_found の値は **true** になっており，どの v も u と隣接していなかったら (この場合，$u \notin S(k)$ である) path_found の値は初期値 **false** のまま変化していない．よって，path_found の値をそのまま path_exist の関数値とすればよい (行 5 の **else** 節)．$p < |S(k-1)|$ の場合には，$v \in S(k-1)$ であるにもかかわらず，行 3.1 において呼び出した member_guess が推測に失敗して停止してしまったために考慮しそこなった v が存在することを意味する．この場合，$u \in S(k)$ かどうかを正しく判定することができないのでアルゴリズムをそこで停止させる．以上の考察により，アルゴリズムが途中で停止することなく path_exist が計算した値は必ず正しい，ということが分かる．関数 path_exist は $O(\log n)$ 領域しか使わない．

最後に，M は $|S(0)|, |S(1)|, \cdots$ を次のように順次計算し，$|S(n-1)|$ を計算結果として出力して終了する．

```
procedure counting_GAP(G, x)
begin
    |S(0)| ← 1;
    for k ← 1 to n − 1 do /* n = |V(G)| */
        |S(k)| ← 0;
        forall u ∈ V(G) do
            if path_exist(G, x, u, k, |S(k − 1)|) then |S(k)| ← |S(k)| + 1 end
        end;
        |S(n − 1)| を出力し，入力 (G, x) を受理して停止する
    end
end
```

$|S(k)|$ を計算するにあたって M は $|S(k-1)|$ だけを記憶しておく (k の値ごとに計算用の領域を繰返し再利用する)．したがって，M が使う領域量は $O(\log n)$ で十分である．こうして (M が途中で停止することなく) 求められた $|S(n-1)|$ が，x から到達可能な頂点の個数に等しいことは明らかであろう ($|S(k)|$ が，x から長さ k 以下の道に沿って到達可能な頂点の個数であることを k に関する帰納法で証明せよ)．すなわち，入力 (G,x) が M に受理されたときの出力は必ず正しい． また，どんな (G,x) に対しても，member_guess が v_i の推測にすべて成功して $p = |S(k-1)|$ となる場合が必ず存在するので，そのとき path_exist も counting_GAP も途中で停止せず，M は (G,x) を受理して正しい最終計算結果を出力する．よって，任意の (G,x) に対し，$|M(G,x)| = 1$ であり $M(G,x)$ の元は「x から到達可能な頂点の個数」である． □

任意の $f(n)$ に対して **DSPACE**$(f(n))$ や **DTIME**$(f(n))$ が補集合をとる演算で閉じていることは容易に示すことができる (読者への問とする) が，非決定性のクラスに対しても同じことが成り立つかどうかは長い間未解決であった．次の定理は **NSPACE**$(f(n))$ についての肯定的な答である†．

定理 4.3 (Immerman-Szelepcsényi の定理) $s(n) \geq \log n$ が領域構成可能関数なら **NSPACE**$(s(n))$ = co-**NSPACE**$(s(n))$ である．

証明 $L \in$ **NSPACE**$(s(n))$ で，L を受理する $s(n)$ 領域限定 NTM を M とする．L の補集合 L^c を受理する $s(n)$ 領域限定 NTM $\bar{M}$ が存在することを示せばよい．定理 4.1 の証明において述べたように，長さ n の入力 x に対して M が取り得る ID の個数はたかだか $c^{\log n + s(n)}$ である (c は定数)．とくに $s(n) \geq \log n$ なら，これは $\leq (c^2)^{s(n)}$ である．よって，補題 4.1 より，入力 x に対する M の様

† この定理は Immerman と Szelepcsényi がほとんど同時に独立に証明した:
N.Immerman, Nondeterministic space is closed under complementation, *SIAM J. Comput.* 17, pp.935-938, 1988.
R.Szelepcsényi, The method of forced enumeration for nondeterministic automata, *Acta Inform.* 26, pp.279-284, 1988.

相グラフ $G_{M,x}$ の到達可能性問題の数え上げ版を解く $O(\log(c^2)^{s(n)}) = O(s(n))$ 領域限定 NTM が存在する．入力 x が与えられたら $\bar{M}$ は $G_{M,x}$ の到達可能性問題を補題 4.1 のように解く (すなわち，counting_GAP($G_{M,x}$, $\mathrm{Init}_{M,x}$) を実行する．ここで，$\mathrm{Init}_{M,x}$ は入力 x に対する M の初期 ID である)．$\bar{M}$ は $G_{M,x}$ の全辺を記憶しておくわけではなく，$(I_1, I_2) \in E(G_{M,x})$ かどうかを知る必要が生じるたびに M の遷移関数を調べる．したがって，$\bar{M}$ が使う領域量は補題 4.1 の実行に必要な領域量 $O(s(n))$ だけで十分である．counting_GAP($G_{M,x}$, $\mathrm{Init}_{M,x}$) を実行する過程で，ある $S(k)$ に受理 ID が入ったら (すなわち M が x を受理したら) $\bar{M}$ は x を拒否して停止する．そうでなく，M が受理 ID に入ることなく $S(n-1)$ の計算が終わったら x を受理する．

$x \in L(M)$ とすると $\mathrm{Init}_{M,x} \vdash_M^k I_f$ となる受理 ID I_f と整数 k が存在する．ゆえに，$I_f \in S(k)$ であり，$\bar{M}$ における拒否の定義より $x \notin L(\bar{M})$ である．$x \notin L(M)$ とすると，初期 ID $\mathrm{Init}_{M,x}$ から到達可能な受理 ID は存在しないから $\bar{M}$ は途中で x を拒否して停止することなく $|S(0)|, |S(1)|, \cdots$ を順次計算し，$S(n-1)$ を計算し終わったところで x を受理して停止する．よって，$L(\bar{M}) = L(M)^c$ である． □

この節を終わるにあたって，帰納的言語と帰納的可算言語との関係について述べておきたい．まず，補集合をとる演算で閉じているかどうかという観点からいうと，

問 4.3 **REC** は補集合で閉じていることを示せ．

であることを容易に示すことができるが (補集合で閉じている他のクラスについては，演習問題 5 参照)，帰納的可算言語の補集合は帰納的言語と次のような密接な関係にある．

定理 4.4 $\mathbf{REC} = \mathbf{RE} \cap \text{co-}\mathbf{RE}$.

証明 $\mathbf{REC} \subseteq \mathbf{RE}$ は自明．また，問 4.3 より，$\mathbf{REC} \subseteq \text{co-}\mathbf{RE}$ である．

逆を示すために $L \in \mathbf{RE} \cap \text{co-}\mathbf{RE}$ とすると，L を受理する DTM M と L^c を受理する DTM M^c が存在する．次のように動作する DTM M' を考える: 入力 x が与えられたら，M' は M と M^c の動作を交互に 1 ステップずつ模倣し，M が x を受理したら M' は x を受理し，M^c が x を受理したら M' は x を拒否する．どんな x も M または M^c に必ず受理されるから，M' は停止性 TM である．よって，$L \in \mathbf{REC}$. □

帰納的言語でない帰納的可算言語が存在することを 4.4 節で示す．このことと定理 4.4 より，**RE** は補集合で閉じていないことが導かれる．

4.2 決定性計算量の階層

直感的には，より多くの計算資源 (時間や計算スペース) を使えば，より難しい問題が解けると考えられる．しかし，線形加速定理や線形領域圧縮定理が示すように，たかだか定数倍程度計算量を増やすだけでは解くことのできる問題の範囲は広がらない．では，どの程度増やせばそれが可能になるのであろうか? この節では，その答を求める．

はじめに，計算量のクラスと帰納的言語の関係について考察する．定義より，任意の $t(n)$ について，$\mathbf{DTIME}(t(n))$ に属す言語は $t(n)$ 時間限定の停止性 DTM に受理されるのだから帰納的言語である．一方，使用する領域量が有限であっても計算がループに陥るような DTM もある．しかし，任意の $s(n)$ について，$s(n)$ 領域限定 DTM の計算がループに陥ったかどうかを $O(c^{s(n)})$ 時間 ($c > 0$ は定数) で判定できる (補題 4.2 参照) から，$\mathbf{DSPACE}(s(n))$ に属す言語も帰納的言語である．だが，次の定理に示すように，帰納的言語の全体は一つの計算量のクラスとはならない．

定理 4.5 $f(n)$ を任意の帰納的関数とする．$\mathbf{DTIME}(f(n))$ にも $\mathbf{DSPACE}(f(n))$ にも属さない帰納的言語 L が存在する．

証明 $\mathbf{DTIME}(f(n))$ の方を証明する ($\mathbf{DSPACE}(f(n))$ に対する証明も同様である)．$f(n)$ は帰納的関数 (したがって，全域関数) であるから，$f(n)$ を計算する (すなわち，長さ n の入力に対し $1^{f(n)}$ を出力テープに書き出す) 停止性 DTM M が存在する．

2.3 節の例 2.5 (4) に述べたように，多テープ DTM をアルファベット $\{0,1\}$ 上の語として符号化する．さらに，$\alpha \in \{0,1\}^*$ が，ある DTM のコードなら，任意の $i \in \mathbf{N}$ に対して $0^i\alpha$ も同じ DTM のコードであるとする．また，$\{0,1\}$ 上の語のうち DTM のコードの形をしていないものは，空集合を受理する DTM のコードであるとみなすことにする．このように考えると，$\{0,1\}^*$ の元を標準順序で並べた $x_1, x_2, \ldots$ は多テープ DTM の列挙 $M_{x_1}, M_{x_2}, \ldots$ である．

$$L = \{x_i \mid M_{x_i} \text{ は } x_i \text{ を } f(|x_i|) \text{ ステップ以内では受理しない}\}$$

と定義する．L を受理する停止性 TM M' が存在することを示そう．M' は M の出力テープを普通のテープとして使う．また，作業用のテープを 1 本余分に持つものとする．長さ n の入力 x_i が与えられたとき，M' はまず入力を余分な作業テープにコピーした後，この入力 x_i に対して M の動作を模倣する．その結果，出力テープに $1^{f(n)}$ が書き出される．次いで，コピーしておいた x_i を入力として M_{x_i} の動作を模倣する．すなわち，x_i を多テープ DTM のコードだと考え，その遷移関数を求め，それに従った動作を行なう．x_i が DTM のコー

ドの形をしていなかったら M' は何もせず入力を受理して停止する．模倣のステップ数が $f(n)$ を越える前に M が入力 x_i を拒否した場合 ($f(n)$ を越えたかどうかは，1 ステップごとに出力テープヘッドを 1 こま動かすことによって判定できる)，または模倣が $f(n)$ ステップを越えてから M が x_i を受理した場合 M' は x_i を受理し，それ以外の場合は拒否する．明らかに，$L(M') = L$ である．

もし $L \in \mathbf{DTIME}(f(n))$ とすると，L を受理する $f(n)$ 時間限定 DTM (M_{x_i} とする) が存在する．もし $x_i \in L(M_{x_i}) = L$ とすると，M_{x_i} は x_i を $f(|x_i|)$ ステップ以内に受理する．これは $x_i \notin L$ を意味し，矛盾である．逆に，$x_i \notin L(M_{x_i})$ とすると M_{x_i} は x_i を受理せず，かつ M_{x_i} は $f(n)$ 時間限定だから，L の定義より $x_i \in L$ となり，これも矛盾である． □

このように，対象となるものを $\text{object}_1, \text{object}_2, \ldots$ と列挙し (定理 4.5 では，$M_{x_1}, M_{x_2}, \ldots$)，この中のどれともどこか 1 点で異なるような対象 (定理 4.5 では，L を受理する M_{x_i}) を示すことによって矛盾を導くという証明法は，集合論の創始者 G.Cantor (1890 年) によって考え出されたもので，**対角線論法** (diagonal argument) と呼ばれている．

命題 3.5 (間隙定理) において，計算量のクラスの間には任意に大きい間隙が存在することを述べた．それでは，どのような $f(n), g(n)$ に対して $\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n))$，あるいは $\mathbf{DSPACE}(f(n)) \subsetneq \mathbf{DSPACE}(g(n))$ となるのであろうか？ 以下で，$f(n)$ と $g(n)$ の関係を具体的に示す．

はじめに，領域量の場合を考察する．

補題 4.2 任意の n に対して $s(n) \geq \log n$ とする．M が $s(n)$ 領域限定 DTM なら，M を模倣する $s(n)$ 領域限定の停止性 DTM M' が存在する．

証明 定理 3.2 より，M は 1 作業テープオフライン DTM と仮定してもよい．定理 4.1 (2) の証明において述べたように，ある定数 d が存在して，M が取り得る ID の個数はたかだか $d^{\log n + s(n)}$ である．$s(n) \geq \log n$ だから $d^{\log n + s(n)} \leq c^{s(n)}$ となる c が存在する．

M' は作業テープを 2 本持ち，1 本は M の模倣に使い，他の 1 本はステップ数をカウントするためのカウンタに使う．カウンタとして使える領域の大きさを M が使う領域量と等しくするために，M が新しいこまを 1 つ使うたびに M' はカウンタ用の領域を 1 こま広げる (こまに印を付ける)．M' は，M の 1 ステップを模倣するたびにカウンタに c 進数として 1 を加える．模倣の途中でカウンタの値がカウンタ用の領域を越えるほど大きくなったら M' は停止して入力を拒否する．このようなことが起こるのはステップ数が $c^{s(n)}$ 以上となっ

た場合だけであり，そのとき M の計算においては少なくとも 1 つ，同じ ID が繰り返している (したがって M の計算はループに陥るので，入力は受理されない)．それ以外の場合，M が入力を受理 (拒否) したら M' も受理 (拒否) する．このようにして M' は M がループに陥ったかどうかを判定でき，任意の入力に対して必ず停止する． □

定理 4.6 (決定性領域量に関する階層定理) $s_1(n) \geq \log n$, $s_2(n) \geq \log n$ で $s_2(n)$ は領域構成可能関数とする．$\inf_{n\to\infty} \dfrac{s_1(n)}{s_2(n)} = 0$ ならば $\mathbf{DSPACE}(s_2(n)) - \mathbf{DSPACE}(s_1(n)) \neq \emptyset$ である．

証明 対角線論法で証明する．定理 4.5 と同じ DTM の列挙を考える．$s_2(n)$ を領域構成する DTM を N_2 とする．入力 $x \in \{0,1\}^*$ ($|x| = n$) に対して，次のように動作する DTM M を考える:

1. はじめに M は N_2 を模倣して $s_2(n)$ 領域に印を付ける．以後，この領域をはみ出るようなことが起こったら M は入力を拒否する．
2. 次いで，M は M_x の模倣を始める．補題 4.2 より，M_x はすべての入力に対して停止すると仮定してよい．M_x が $s(n)$ 領域限定で M_x のテープ記号の個数が t ならば，この模倣に使う領域量は $\lceil \log_2 t \rceil s(n)$ あれば十分である (M のテープアルファベットは $\{0,1\}$ だから M_x のテープ記号を 2 進数で符号化する必要があるため)．
3. 模倣が $s_2(n)$ 領域以内で完了し M_x が入力 x を拒否したら M は x を受理し，それ以外の場合は拒否する．

明らかに M は $s_2(n)$ 領域限定であるから $L(M) \in \mathbf{DSPACE}(s_2(n))$ である．$L(M) \notin \mathbf{DSPACE}(s_1(n))$ であることを示そう．もし $L(M) \in \mathbf{DSPACE}(s_1(n))$ であるとすると，DTM の列挙 $M_{x_1}, M_{x_2}, \ldots$ の中に，$L(M)$ を受理する $s_1(n)$ 領域限定の DTM M_x が存在する．M_x のテープ記号の個数を t とする．補題 4.2 より，M_x は停止性 TM であると仮定してよい．$\inf_{n\to\infty} \dfrac{s_1(n)}{s_2(n)} = 0$ だから，十分大きい n_0 を選ぶと

$$(*) \quad \lceil \log_2 t \rceil s_1(n_0) < s_2(n_0)$$

とすることができる．$|00\cdots 0x| = n_0$ となるような $x' = 00\cdots 0x$ を選ぶと，M_x と $M_{x'}$ は同じ TM だから $L(M_{x'}) = L(M_x)$ である．M に x' を入力したとき，M_x は $s_1(n)$ 領域限定だから $(*)$ より，M は $M_{x'}$ の模倣を完成させることができる．$M_{x'}$ は停止性 TM だから，M の定義の仕方より

$$x' \notin L(M_{x'}) = L(M_x) \iff x' \in L(M)$$

が成り立つ．これは $L(M_x) \neq L(M)$ を意味し，矛盾である． □

時間量の場合も定理 4.6 とほとんど同じ論法で次の定理を示すことができる．領域量の場合より弱い結果となっているのは，多テープ TM を 2 テープで模倣するのにかかる時間 (定理 3.5) のためである．

定理 4.7 (決定性時間量に関する階層定理) $t_2(n)$ を時間構成可能関数とする．$\inf_{n\to\infty} \dfrac{t_1(n)\log t_1(n)}{t_2(n)} = 0$ ならば $\mathbf{DTIME}(t_2(n)) - \mathbf{DTIME}(t_1(n)) \neq \emptyset$ である．

証明 証明は 定理 4.6 とほとんど同じであるから，注意点だけを述べ，詳細は読者への演習問題とする．M_x のテープ本数は x によっていろいろであるからといって M のテープ本数をいろいろ変えるわけにはいかないので，M は 2 テープであるとする．このため，M が M_x を模倣するのに $O(t_1(n)\log t_1(n))$ 時間かかってしまう (定理 3.5)．M は M_x の模倣と同時に，$t_2(n)$ を時間構成する DTM の模倣を並行して行なう．$t_2(n)$ ステップ以内に M_x が停止して x を拒否したら M は x を受理する． □

例 4.1 (1) $\inf_{n\to\infty} \dfrac{s_1(n)}{s_2(n)} = 0$ かつ任意の n に対して $s_1(n) \leq s_2(n)$ ならば $\mathbf{DSPACE}(s_1(n)) \subsetneq \mathbf{DSPACE}(s_2(n))$ である．例えば，任意の実数 $\varepsilon > 0$ に対して $\mathbf{DSPACE}(\log n) \subsetneq \mathbf{DSPACE}(n^\varepsilon)$ である．

(2) $\varepsilon > 1$ なら $\inf_{n\to\infty} \dfrac{2^n \log 2^n}{n^\varepsilon 2^n} = 0$ かつ任意の n に対して $2^n \leq n^\varepsilon 2^n$ だから $\mathbf{DTIME}(2^n) \subsetneq \mathbf{DTIME}(n^\varepsilon 2^n)$ である． □

系 4.3 (1) $\mathbf{DL} \subseteq \mathbf{NL} \subsetneq \mathbf{PSPACE} \subsetneq \mathbf{EXPSPACE}$.
(2) $\mathbf{P} \subsetneq \mathbf{DEXP}$, $\mathbf{NP} \subsetneq \mathbf{EXPSPACE}$.

証明 (1) Savitch の定理および定理 4.6 により $\mathbf{NL} \subsetneq \mathbf{PSPACE}$ である．また，定理 4.6 より $\mathbf{PSPACE} \subsetneq \mathbf{EXPSPACE}$.
(2) 前者は定理 4.7 より，後者は $\mathbf{NP} \subseteq \mathbf{PSPACE}$ と (1) より導かれる． □

$\mathbf{DL} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{DEXP} \subseteq \mathbf{NEXP} \subseteq \mathbf{EXPSPACE}$ であるが，これらのクラスの間の関係としては現在のところ，系 4.3 以上のことは何も分かっていない．

問 4.4 次のクラスの間の包含関係は？
(a) $\mathbf{DTIME}(2^n)$ と $\mathbf{DSPACE}(3^n)$ (b) $\mathbf{NSPACE}(2^n)$ と $\mathbf{DSPACE}(5^n)$
(c) $\mathbf{DSPACE}(\sqrt{n})$ と $\mathbf{NTIME}(\log n)$

4.3 詰め込み論法と非決定性計算量の階層

定理 4.6，定理 4.7の論法で非決定性計算量の階層定理を証明しようとすると，対角線論法を適用するために NTM を DTM で模倣する必要があり，そのため，領域については $\inf_{n\to\infty} \dfrac{s_1(n)^2}{s_2(n)} = 0$，時間については $\inf_{n\to\infty} \dfrac{c^{t_1(n)}}{t_2(n)} = 0$ という，あまりよくない結果しか得られない．そこで，以下に述べるような**詰め込み論法** (padding argument) を用いて計算量の低いクラスの間の包含関係を計算量がより高いクラスの間の包含関係に変換することにより，より精密な非決定性の階層が得られる．詰め込み論法というのは，計算量の高い言語の元である各語の末尾に “詰め物” をすることによって長さを引き伸ばし，その結果 (長さが長くなるだけで本来の計算量は変わらないが，長くなった分だけ相対的に計算量が小さくなる) もともとの言語を計算量の小さい言語に変換するという証明手法である．

$f : \mathbf{N} \to \mathbf{N}$ を $f(n) \geq n$ なる関数とする．アルファベット Σ 上の言語 L に対し，言語 $L[f]$ を次のように定義する:

$$L[f] = \{x\#^{f(|x|)-|x|} \mid x \in L\}.$$

ただし，$\#$ は Σ にない文字とする．

定理 4.8 $f(n) \geq n$，$s(n) \geq \log n$ で，$f(n)$ は $O(\log f(n))$-領域構成可能であるとする．このとき，任意の言語 $L \subseteq \Sigma^*$ に対して次の (i), (ii) が成り立つ．
(i) $L \in \mathbf{DSPACE}(s(f(n))) \iff L[f] \in \mathbf{DSPACE}(s(n))$.
(ii) $L \in \mathbf{NSPACE}(s(f(n))) \iff L[f] \in \mathbf{NSPACE}(s(n))$.

証明 (i) も (ii) も証明は同様なので (i) について示す．

($\Longrightarrow$) $L \in \mathbf{DSPACE}(s(f(n)))$ とする．L を受理する $s(f(n))$ 領域限定のオフライン DTM M_1 が存在する．また，長さ n の任意の入力に対し tally 表現 $1^{f(n)}$ を生成する $O(\log f(n))$ 領域限定のオフライン DTM M_2 が存在する．明らかに，M_2 は $1^{f(n)}$ の代わりに値 $f(n)$ の 2 進数表現を $\log f(n)$ 領域で生成するように直すことができる (したがって，M_2 は $\log f(n)$ 領域限定)．$L[f]$ を受理するオフライン DTM M_3 を次のように構成する:

1. 長さ n の入力 w が与えられたとき (入力文字列 w の前後の終止符 ¢, \$ は省略してある), M_3 はまず, この入力が $x\#^{n-m}$, $x \in \Sigma^*$, $|x| = m$, の形をしているかどうかを調べ, そうでなかったら入力を拒否して停止する. このステップで使う領域量は $O(1)$ である.
2. 次に, M_3 は作業テープ 1 の $\log n$ 個のます目に印を付ける. 関数 $\log n$ は領域構成可能だから, これは $O(\log n)$ 領域でできる.
3. つづけて M_3 は M_2 を使って作業テープ 1 の上に $f(m)$ の 2 進数表現を計算して書き出す. ただし, ステップ 2 で印を付けた領域を計算の途中ではみ出たら入力を拒否する. 計算が終了したら $f(m) = n$ であるかどうか調べ, そうでなかったら入力を拒否する. このステップで使う領域量は $O(\log n)$ である.
4. 最後に, M_3 は M_1 を使って $x \in L$ かどうかを調べ, そうであったら入力を受理, そうでなかったら拒否し, 停止する. このステップで使う領域量は $O(s(f(m))) = O(s(n))$ である.

$s(n) \geq \log n$ であるから M_3 は $O(s(n))$ 領域限定である. w が M_3 に受理されるなら明らかに $w \in L[f]$ である. 逆に, $w \in L[f]$ のとき, w がステップ 1, 2 で拒否されることがないのは明らかである. ステップ 3 において $f(m)$ を計算する際に $\log n$ 領域をはみ出ると M_3 は停止してしまうが, $w \in L[f]$ ならば $f(m) = n$ であり, M_2 は $\log f(m)$ 領域限定であるから, そのようなことは起こらず w は M_3 に受理される. よって, $L(M_3) = L[f]$ である.

($\Longleftarrow$) $L[f] \in \mathbf{DSPACE}(s(n))$ とする. $L[f]$ を受理する $s(n)$ 領域限定のオフライン DTM を M'_1 とする. L を受理するオフライン DTM M'_2 を次のように構成する:

1. x ($|x| = n$) を M'_1 への入力とする. まず, M'_2 は M_2 を使って $f(n)$ の 2 進数表現を計算し, 値 $f(n) - n$ を作業テープ 1 (2 進カウンタ) に書き込む. このステップで使う領域量は $O(\log f(n))$ である.
2. 次に, M'_2 は M'_1 の動作を模倣する. その際, 入力ヘッドが x の範囲にあるときは M'_1 をそのまま模倣する. 入力ヘッドが x の右の終止符 \$ を読んでいるときは, M'_2 は入力記号として # を読んでいるものとして M'_1 を模倣する. このとき同時に作業テープ 2 をカウンタとして使い, x の右端からの距離 m を保持しておく (すなわち, M'_1 の入力ヘッドが右に動くときはカウンタに 1 加え, 左に動くときは 1 引く). $1 \leq m \leq f(n) - n$ である間は M'_2 は # を読んでいるものとして M'_1 の模倣を行なう. M'_1 が入力を受理したときに $m = f(n) - n$ であったら (このとき M'_1 は

$x\#^{f(n)-n}$ を受理している) M_2' は入力 x を受理する．M_1' が入力を受理したとき $m \neq f(n)-n$ であった場合，M_1' が入力を拒否した場合，模倣の途中で $m > f(n)-n$ となった場合は，いずれも M_2' は入力を拒否する．M_1' は $s(n)$ 領域限定であるから，このステップの領域量は $O(\max\{\log(f(n)-n), s(f(n))\})$ である．

$s(f(n)) \geq \log f(n)$ であるから M_2' は $O(s(f(n)))$ 領域限定である．M_2' の構成の仕方より，$x \in L(M_2') \iff x\#^{f(|x|)-|x|} \in L(M_1')$ が成り立つ．$L(M_1') = L[f]$ であるから，後者が成り立つ必要十分条件は $x \in L$ である．すなわち，$L(M_2') = L$. □

系 4.4 (領域量に関する変換補題) $f(n)$ は $O(\log f(n))$-領域構成可能関数で，$s_1(n) \geq \log n$, $s_2(n) \geq \log n$, $f(n) \geq n$ とする．このとき，$\mathbf{DSPACE}(s_1(n)) \subseteq \mathbf{DSPACE}(s_2(n))$ ならば $\mathbf{DSPACE}(s_1(f(n))) \subseteq \mathbf{DSPACE}(s_2(f(n)))$ である．同様なことが **NSPACE** に対しても成り立つ．

証明 $L \in \mathbf{DSPACE}(s_1(f(n)))$ とする．定理 4.8 より $L[f] \in \mathbf{DSPACE}(s_1(n))$. よって $L[f] \in \mathbf{DSPACE}(s_2(n))$. 再び定理 4.8 により $L \in \mathbf{DSPACE}(s_2(f(n)))$. **NSPACE** についても同様． □

時間量についても領域量と同様な変換補題が成り立つ．次の定理の証明は定理 4.8 と同様である．

定理 4.9 $f(n) \geq n$, $t(n) \geq n$ で，$f(n)$ は時間構成可能関数とする．このとき，任意の言語 L に対して次の (i), (ii) が成り立つ．

(i) $L \in \mathbf{DTIME}(t(f(n))) \iff L[f] \in \mathbf{DTIME}(t(n))$.

(ii) $L \in \mathbf{NTIME}(t(f(n))) \iff L[f] \in \mathbf{NTIME}(t(n))$.

系 4.5 (時間量に関する変換補題) $f(n)$ は時間構成可能関数で，$f(n) \geq n$, $t_1(n) \geq n, t_2(n) \geq n$ とする．このとき，$\mathbf{DTIME}(t_1(n)) \subseteq \mathbf{DTIME}(t_2(n))$ ならば $\mathbf{DTIME}(t_1(f(n))) \subseteq \mathbf{DTIME}(t_2(f(n)))$ である．同様なことが **NTIME** に対しても成り立つ．

例 4.2 (1) $\mathbf{DTIME}(2^n) \subsetneq \mathbf{DTIME}(n2^n)$ であることを示そう．

$$\inf_{n\to\infty} \frac{2^n \log 2^n}{n2^n} = 1$$

であるから，この結果は DTM の階層定理 (定理 4.7) からは直接得られないものである．さて，

$$\mathbf{DTIME}(n2^n) \subseteq \mathbf{DTIME}(2^n) \tag{a}$$

と仮定して矛盾を導こう．$t_1(n) = n2^n$, $t_2(n) = 2^n$, $f(n) = 2^n$ として系 4.5 を適用すると

$$\mathbf{DTIME}(2^n 2^{2^n}) \subseteq \mathbf{DTIME}(2^{2^n}) \tag{b}$$

を得る．同様に，$f(n) = n + 2^n$ とすると

$$\mathbf{DTIME}((n+2^n)2^{n+2^n}) \subseteq \mathbf{DTIME}(2^{n+2^n}) \tag{c}$$

を得る．したがって，(b), (c) より

$$\mathbf{DTIME}((n+2^n)2^{n+2^n}) \subseteq \mathbf{DTIME}(2^{2^n}) \tag{d}$$

を得るが，$\inf_{n\to\infty} \frac{2^{2^n} \log 2^{2^n}}{(n+2^n)2^{n+2^n}} = 0$ であるから，(d) は階層定理に矛盾する．よって，$\mathbf{DTIME}(n2^n) \not\subseteq \mathbf{DTIME}(2^n)$. 一方 $\mathbf{DTIME}(2^n) \subseteq \mathbf{DTIME}(n2^n)$ であるから，$\mathbf{DTIME}(n2^n) \subsetneq \mathbf{DTIME}(2^n)$.

(2) $\mathbf{DSPACE}(n) \subseteq \mathbf{P} \Longleftrightarrow \mathbf{P} = \mathbf{PSPACE}$ を示そう．

$\mathbf{P} = \mathbf{PSPACE}$ ならば明らかに $\mathbf{DSPACE}(n) \subseteq \mathbf{P}$ である．

$\mathbf{DSPACE}(n) \subseteq \mathbf{P}$ とし，$L \in \mathbf{PSPACE}$ とする．ある k が存在して $L \in \mathbf{DSPACE}(n^k)$ である．$f(n) = n^k$, $s(n) = n$ として定理 4.8 を適用すると $L[f] \in \mathbf{DSPACE}(n)$. 仮定より $\mathbf{DSPACE}(n) \subseteq \mathbf{P}$ であるから，$L[f] \in \mathbf{DTIME}(n^\ell)$ となる ℓ が存在する．$f(n) = n^k$, $t(n) = n^\ell$ として定理 4.9 を適用すると $L \in \mathbf{DTIME}(n^{k\ell})$. よって $L \in \mathbf{P}$ であり，$\mathbf{PSPACE} \subseteq \mathbf{P}$ が証明された．$\mathbf{P} \subseteq \mathbf{PSPACE}$ であるから $\mathbf{P} = \mathbf{PSPACE}$ が結論される．

(3) $\mathbf{DL} = \mathbf{NL}$ ならば，$g(n) \geq \log_2 n$ なる任意の領域構成可能関数 $g(n)$ に対し $\mathbf{DSPACE}(g(n)) = \mathbf{NSPACE}(g(n))$ である．

これを証明しよう．$f(n) = 2^{g(n)}$ とおくと $f(n)$ は $g(n)$-領域構成可能である．すなわち，$f(n)$ は $\log f(n)$-領域構成可能である．$g(n) \geq \log_2 n$ だから $f(n) \geq n$ である．$s(n) = \log n$ として定理 4.8 を適用する．$L \in \mathbf{NSPACE}(g(n))$ とすると $L[f] \in \mathbf{NSPACE}(\log n) = \mathbf{NL}$ である．一方，$\mathbf{NL} = \mathbf{DL}$ という仮定より，$L[f] \in \mathbf{DL} = \mathbf{DSPACE}(\log n)$ である．再び定理 4.8 より，$L \in \mathbf{DSPACE}(g(n))$ が得られる．　□

問 4.5 (1) $\mathbf{P} \neq \mathbf{DSPACE}(n)$ であることを示せ．
(2) $\mathbf{NTIME}(n) \subseteq \mathbf{P} \Longleftrightarrow \mathbf{P} = \mathbf{NP}$ を示せ．

例 4.2 でも見たように，詰め込み論法によって，すでに得られている計算量のクラスの階層をさらに精細にすることができる．非決定性計算量に関する次

の階層定理はその典型的な例である[‡].

定理 4.10 (1) 任意の実数 r_1, r_2, $0 \le r_1 < r_2$ に対して $\mathbf{NSPACE}(n^{r_1}) \subsetneq \mathbf{NSPACE}(n^{r_2})$.

(2) 任意の実数 r_1, r_2, $1 \le r_1 < r_2$ に対して $\mathbf{NTIME}(n^{r_1}) \subsetneq \mathbf{NTIME}(n^{r_2})$.

証明 (1) $r_1 \le s/t$, $(s+1)/t \le r_2$ を満たす正整数 s,t が存在するので，任意の正整数 s,t に対して

$$\mathbf{NSPACE}(n^{s/t}) \subsetneq \mathbf{NSPACE}(n^{(s+1)/t}) \tag{i}$$

が成り立つことを示せばよい．明らかに $\mathbf{NSPACE}(n^{s/t}) \subseteq \mathbf{NSPACE}(n^{(s+1)/t})$ であるから，もし (i) が成り立たないとすると

$$\mathbf{NSPACE}(n^{(s+1)/t}) \subseteq \mathbf{NSPACE}(n^{s/t})$$

が成り立つ．$f(n) = n^{(s+i)t}$ として系 4.4 を適用すると

$$\mathbf{NSPACE}(n^{(s+1)(s+i)}) \subseteq \mathbf{NSPACE}(n^{s(s+i)}) \tag{ii}$$

を得る．ここに，$i = 0, 1, \ldots, s$. 一方，$i \ge 1$ ならば $s(s+i) \le (s+1)(s+i-1)$ であるから

$$\mathbf{NSPACE}(n^{s(s+i)}) \subseteq \mathbf{NSPACE}(n^{(s+1)(s+i-1)}) \tag{iii}$$

が成り立つ．(ii), (iii) を繰返し使うと，

$$\begin{array}{rlcl} & \mathbf{NSPACE}(n^{(s+1)(2s)}) & \subseteq & \mathbf{NSPACE}(n^{s(2s)}) \\ \subseteq & \mathbf{NSPACE}(n^{(s+1)(2s-1)}) & \subseteq & \mathbf{NSPACE}(n^{s(2s-1)}) \\ \subseteq & \cdots & \subseteq & \mathbf{NSPACE}(n^{s(s+1)}) \\ = & \mathbf{NSPACE}(n^{(s+1)s}) & \subseteq & \mathbf{NSPACE}(n^{s^2}). \end{array}$$

が得られる．一方，Savitch の定理 (定理 4.2) により

$$\mathbf{NSPACE}(n^{s^2}) \subseteq \mathbf{DSPACE}(n^{2s^2})$$

であり，決定性領域量に対する階層定理 (定理 4.6) より

$$\mathbf{DSPACE}(n^{2s^2}) \subsetneq \mathbf{DSPACE}(n^{2s^2+2s})$$

である．明らかに

$$\mathbf{DSPACE}(n^{2s^2+2s}) \subseteq \mathbf{NSPACE}(n^{2s^2+2s})$$

[‡] O.H.Ibarra, A note concerning nondeterministic tape complexity, *J. Assoc. Comput. Mach.* 19, pp.608-612, 1972.

S.A.Cook, A hierarchy for nondeterministic time complexity, *J. Comput. Sys. Sci.* 7, pp.343-353, 1973.

J.I.Seiferas, M.J.Fischer and A.R.Meyer, Separating nondeterministic time complexity classes, *J. Assoc. Comput. Mach.* 25, pp.146-167, 1978.

であるから，これらの不等式を結合すると

$$\mathbf{NSPACE}(n^{2s^2+2s}) \subsetneq \mathbf{NSPACE}(n^{2s^2+2s})$$

が得られ，これは矛盾である．

時間量に関しては Savitch の定理に相当するものがないので，(2) の証明は (1) ほど簡単ではない．脚注に示した Cook および Seiferas-Fischer-Meyer の論文を参照せよ．　□

4.4　計算不能問題

yes/no の答を求める問題のことを決定問題ということは，すでに述べた．2.3 節で，われわれはこの種の "問題" を適当なアルファベット上の "言語" に符号化して，TM による言語の認識問題として定式化した．A を決定問題とし，Σ を符号用のアルファベット，$L_A \subseteq \Sigma^*$ を A のコードとしよう．Σ 上の "語" x は A の 1 つの具体例 (**インスタンス** (instance) という) を表わす．言語 L_A を受理する停止性 TM M が存在するならば，M は A を解くアルゴリズムと考えることができる (x の答が yes $\Longleftrightarrow$ x が M に受理される)．このとき，問題 A (言語 L_A) は**計算可能** (computable) であるとか帰納的 (recursive) であるという．とくに，決定問題の場合には**決定可能** (decidable) であるともいう．例えば，言語 L と語 x が与えられたとき $x \in L$ か否かを判定する問題を L に対する**所属問題** (帰属問題，membership problem) というが，帰納的言語とは所属問題が決定可能な言語のことに他ならない．

計算可能 (決定可能) でない問題は**計算不能** (**決定不能** (undecidable)) であるという．この節の主題は，決定不能な問題が存在することを示すことである．計算量のクラスはどれも帰納的言語のクラスであることに注意する．したがって，決定不能な問題はどの計算量のクラスにも属さない．

L を受理する (任意の入力に対して必ずしも停止するとは限らない) TM が存在するとき L を帰納的可算言語と呼んだ (2.5 節)．はじめに，帰納的可算でない言語が存在することを対角線論法で示す．

2.3 節で，$\{0,1\}$ を入力アルファベットとする TM を $\{0,1\}$ 上の語 (2 進数) として符号化した．この符号化のもとでは TM を表わさない 2 進数が存在するが，そのような 2 進数は空集合を受理する TM であると考えることにすると，TM と 2 進数は 1 対 1 に対応する．このような仮定のもとで，2 進数 $x \in \{0,1\}^*$ が表わす TM を M_x で表わす．$\{0,1\}$ 上の言語 L_D を次のように定義する ('D' は diagonal (対角線) の頭文字):

$$L_D = \{x \in \{0,1\}^* \mid x \notin L(M_x)\}.$$

補題 4.3 L_D は帰納的可算でない.

証明 L_D が帰納的可算だとすると，L_D を受理する TM M_x が存在する．このとき，$x \in L_D \Longleftrightarrow x \notin L(M_x) = L_D$ となり矛盾である． □

次に，$\{0,1\}$ 上の言語 L_U を

$$L_U = \{\overline{M}\#x \mid x \in L(M)\}$$

と定義する．ただし，M は $\{0,1\}$ を入力アルファベットとする単テープ TM で，$\overline{M}$ はその符号化 (したがって，$M_{\overline{M}} = M$)，$x \in \{0,1\}^*$ は M への入力，$\#$ は 0, 1 と異なる記号である．言語 L_U のことを**万能言語** (universal language) と呼ぶ．L_U は，「TM M に対する所属問題 (TM M と入力 x が与えられたとき，M が x を受理するか否かを判定せよ: 以後，この問題を **TM-MEMBER** で表わす)」を符号化したものである．

定理 4.11 L_U は帰納的可算であるが，帰納的ではない．

証明 まず，帰納的可算であることを証明しよう．L_U を受理する TM M_U を次のように構成する．M_U は 3 本のテープを持ち，テープアルファベットは $\{0, 1, \#, \flat\}$ である．入力は第 1 テープに与えられる．

1. まず最初に，M_U は入力が $\overline{M}\#x$, $x \in \{0,1\}^*$ なる形をしているかどうかをチェックし，そうでなかったら拒否して停止する．
2. 入力が $\overline{M}\#x$ の形をしていたら，M_U は x を第 2 テープにコピーし (ただし，$'0' \to 110 = \overline{a_2}$, $'1' \to 1110 = \overline{a_3}$ のように符号化する)，第 1 テープには M のコード $\overline{M} = 000\overline{\alpha_1}00\overline{\alpha_2}00\cdots00\overline{\alpha_t}000$ だけを残す．各 $\overline{\alpha_?}$ は $1^i01^j01^k01^\ell01^{d+2}$ なる形をしており，これは遷移関数 $\delta(q_i, a_j) = (q_k, a_\ell, d)$ を表わしている．以後，第 2 テープを M のテープの代りに使って M の模倣を行なう．ただし，M のテープ記号のうち $\flat = a_1$ 以外の各記号 a_i はコード $\overline{a_i} = 1^i0$ で表わし，$\flat$ はコード $\overline{a_1} = 10$ ではなく $\flat$ 自身で表わす．M のヘッドが a_i を指しているとき，M_U の第 2 テープヘッドは対応するコード 1^i0 の左端の 1 を指すようにする (このことを，「第 2 テープヘッドは 1^i0 を指している」ということにする)．第 3 テープには M の状態を 1^i なる形 (q_i を表わす) で保持する．M の初期状態は q_1，受理状態は q_2 だけと仮定する (例 2.5 (4) 参照)．
3. テープ 3 の内容が 11 (q_2を表わす) だったら入力を受理して停止する．

4. 第 2 テープヘッドが 1^j0 を指していて，第 3 テープの内容が 1^i だったら，M_U は第 1 テープが $1^i01^j01^k01^\ell01^{d+2}$ なる形の部分語を含んでいるかどうかを調べる．もし含まれていたら，第 2 テープの 1^j0 を $1^\ell0$ に書き換え，$d=-1,0,1$ に従って第 2 テープヘッドを動かし，第 3 テープの内容を 1^k に書き換える．3 へ戻る．もし，第 1 テープがそのような部分語を含んでいなかったら，入力を拒否して停止する．

構成の仕方より明らかに，M_U が $\overline{M}\#x$ を受理する必要十分条件は M が x を受理することである．すなわち，$L(M_U)=L_U$.

上記で構成した TM M_U は，入力された任意の TM のコードを解読して，その動作を模倣する TM である．この意味で，M_U のことを**万能チューリングマシン** (universal TM) という．

次に，L_U が帰納的ではないことを証明しよう．L_U を受理する停止性 TM M が存在したとする．次のように動作する TM M' を考える:

1. 入力 x が $\{0,1\}^*$ の元でなかったら拒否する．$x\in\{0,1\}^*$ だったら x を $x\#x$ に書き換える．
2. $x\#x$ を入力として TM M を模倣する．M が $x\#x$ を受理したら M' は拒否し，M が $x\#x$ を拒否したら M' は受理する．

M は停止性 TM であるから M' も停止性 TM である．構成の仕方より，

$$x\in L(M') \iff x\#x\notin L(M)=L_U \iff x\notin L(M_x) \iff x\in L_D$$

であり，M' は L_D を受理する停止性 TM である．これは補題 4.3 に矛盾する．
□

定理 4.11 では「TM M は x を受理するか？」という問題が決定不能であることを証明したが，実は，

- TM M に空語 ε を入力したとき停止するか？ (TM の空語停止問題)
- TM M が受理する集合は空集合か？
- TM M が受理する集合は無限集合か？

など，TM の受理集合に関する問題は自明なもの (例えば，「TM M が受理する集合は帰納的可算集合か？」という問題の答はつねに yes である) 以外は決定不能である．それを述べたのが次の Rice の定理である．

ある性質 P を満たす，アルファベット $\{0,1\}$ 上の帰納的可算言語の全体を [P] で表わすことにする:

$$[\mathrm{P}]=\{L(M)\mid M\text{は TM, } L(M)\text{ は性質 }P\text{を満たす}\}.$$

[P] のことを，"帰納的可算言語に関する**性質**" P と呼ぶ．例えば，「有限である」という性質とは $\{L \mid L$ は有限な帰納的可算言語$\}$ のことである．帰納的可算言語 (TM M) に対して性質 P が成り立つかどうかを判定する問題は，言語

$$L_{[P]} = \{\overline{M} \mid L(M) \in [\mathrm{P}]\}$$

に対する所属問題に他ならない．すなわち，前者が決定可能であること (このとき，性質 P は決定可能であるという) と，後者が帰納的言語であることとは同値である．[P] $= \emptyset$ または [P] $=$ 帰納的可算言語の全体，であるとき [P] は**自明**であるという．

定理 4.12 (Rice の定理) [P] を帰納的可算言語に関する性質とする．このとき，P が決定可能である必要十分条件は [P] が自明なことである．

証明 ($\Longleftarrow$) [P] $= \emptyset$ なら P はつねに成り立たず，[P] が帰納的可算言語の全体と等しいならば P はつねに成り立つ．いずれにしても P は決定可能である．

($\Longrightarrow$) [P] が自明でなく，P が決定可能であるとする．$\emptyset \notin$ [P] と仮定しても一般性を失わない ($\emptyset \in$ [P] だったら [P] の代わりに $[\mathrm{P}]^c$ を考えればよい)．性質 P を満たす帰納的可算言語 $L \in$ [P] が存在する．L を受理する TM を M_L とする．P は決定可能だから $L_{[P]}$ は帰納的言語である．よって，$L_{[P]}$ を受理する停止性 TM $M_{[P]}$ が存在する．$M_L, M_{[P]}$ を使えば，L_U を受理する停止性 TM M_{L_U} が構成できることを以下で示す．

まず，入力 $\overline{M}\#x$ に対して TM $M_{M,x}$ (のコード $\overline{M_{M,x}}$) を出力する変換機 (すなわち，出力テープを持つ決定性 TM) が存在することに注意する (この変換機を N とする．N は任意の入力に対して必ず停止するように構成できる)．ただし，$M_{M,x}$ は次のように動作する TM である (図 4.1)．最初，$M_{M,x}$ は入力 w と無関係に，x を入力として M を模倣する．M が x を受理しなければ $M_{M,x}$ も w を受理しない．M が x を受理したら，$M_{M,x}$ は w を入力として M_L の模倣を開始する．M_L が w を受理したら $M_{M,x}$ も w を受理する．したがって，

$$(*) \quad x \in L(M) \Longrightarrow L(M_{M,x}) = L, \quad x \notin L(M) \Longrightarrow L(M_{M,x}) = \emptyset$$

である．

M_{L_U} を次のように構成する．

1. はじめに，入力が $\overline{M}\#x$ の形をしているかどうかを調べ，そうでなかったら拒否して停止する．
2. 入力が $\overline{M}\#x$ の形をしていたら，上述の変換機 N によって $\overline{M_{M,x}}$ を生成する．

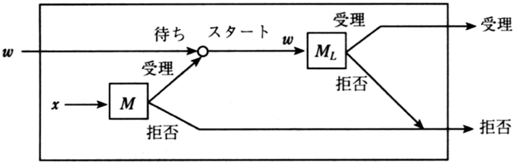


図 **4.1** TM $M_{M,x}$ の構成

3. $\overline{M_{M,x}}$ を入力として $M_{[P]}$ を模倣し，$M_{[P]}$ が $\overline{M_{M,x}}$ を受理したら (すなわち，$L(M_{M,x}) \in [\mathrm{P}]$ だったら) M_{L_U} は入力 $\overline{M}\#x$ を受理し，$M_{[P]}$ が $\overline{M_{M,x}}$ を拒否したら M_{L_U} は $\overline{M}\#x$ を拒否する．

$\overline{M}\#x \in L(M_{L_U})$ とすると $L(M_{M,x}) \in [\mathrm{P}]$ であり，$\emptyset \notin [\mathrm{P}]$ であったから (*) より $x \in L(M)$ である．逆に，$\overline{M}\#x \notin L(M_{L_U})$ とすると $L(M_{M,x}) \notin [\mathrm{P}]$ であり，$L \in [\mathrm{P}]$ だから $L(M_{M,x}) \neq L$ である．したがって，(*) より $x \notin L(M)$ である．よって，$L(M_{L_U}) = L_U$ である．また，N も $M_{[P]}$ も停止性 TM であるから，M_{L_U} は停止性 TM である．これは，定理 4.11 に矛盾する． □

問 4.6 次の問題はいずれも決定不能であることを示せ．
(1) TM の停止問題 (ヒント: TM の空語停止問題から導け)
(2) 変換機 M, 語 x, y に対し，$M(x) = y$ か？

以後，TM の停止問題を **TM-HALT** で表わす．

最後に，これまでに登場した計算量のクラスの間の関係を，図 4.2に示しておこう．太線は真の包含関係を表わす．● はそのクラスに属す言語の例を示す．

演 習 問 題 4

1 次のことを示せ．
(a) $\mathbf{P} \subseteq \text{co-}\mathbf{NP}$.　(b) $\mathbf{NP} \neq \text{co-}\mathbf{NP} \Longrightarrow \mathbf{P} \neq \mathbf{NP}$.

2 AnBn=$\{a^n b^n \mid n \geq 0\} \in \mathbf{DL} - \mathbf{REG}$ を示せ．これより，$\mathbf{REG} \subsetneq \mathbf{DL}$ である (注: このことは，定理 4.6 からは導かれない)．

3 定理 4.2 の証明と同じ論法で，グラフの到達可能性問題を領域量 $O((\log_2 n)^2)$ で解くアルゴリズムが存在することを示せ．このことを使い，定理 4.3 の証明と同じ論法で，定理 4.2 を証明せよ．

4 $O(n)$ 領域限定 NTM (DTM) のことを非決定性 (決定性) **線形有界オートマトン**

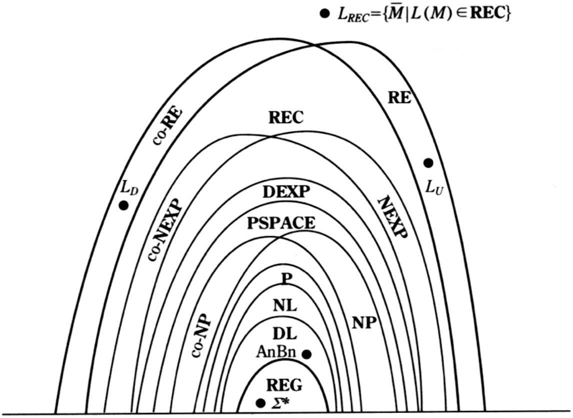


図 **4.2** 計算量のクラスの間の包含関係

(LBA = linear bounded automaton) ともいう．非決定性 LBA が受理する言語 (ε を含むものを除く) のクラスは文脈依存言語のクラスと等しいことが知られている．L が文脈依存言語なら L を受理する非決定性 LBA が存在することを示せ*.

5 文脈依存言語のクラス (= **NSPACE**(n)) が補集合で閉じているか否かは長い間未解決だった有名問題の一つであったが，Immerman の定理で肯定的に解決された.

(1) **DSPACE**(n) も補集合で閉じていることを示せ．これが示されたからといって LBA 問題が肯定的に解かれたことにはならない.

(2) **NSPACE**(1) すなわち正則集合のクラスも補集合で閉じていることを示せ (3 章の演習問題 7 参照).

(3) 任意の $t(n)$ に対して **DTIME**$(t(n))$ および **P** は補集合で閉じていることを示せ．**NTIME**$(t(n))$ や **NP** については未知である (5 章).

6 $m, p, q \geq 1$ を任意の整数とする．次のことを示せ.

(1) $\mathbf{DSPACE}((\log n)^{m+q/(p+1)}) \subsetneq \mathbf{DSPACE}((\log n)^{m+q/p})$,

$\mathbf{DTIME}(n^m \log n) \subsetneq \mathbf{DTIME}(n^{m+1})$.

(2) $\mathbf{NSPACE}((\log n)^{m+q/(p+1)}) \subsetneq \mathbf{NSPACE}((\log n)^{m+q/p})$,

$\mathbf{NTIME}(n^m \log n) \subsetneq \mathbf{NTIME}(n^{m+1})$.

7 詰め込み論法によって次のことを示せ.

(1) **P** = **NP** なら **DEXP** = **NEXP**.

(2) **NSPACE**$(n) \subseteq$ **NP** なら **NP** = **PSPACE**. これから，**NSPACE**$(n) \neq$ **NP**

*S.-Y.Kuroda, Classes of languages and linear bounded automata, *Inform. Contr.* 7, pp.207-223, 1964.

であることを導け.

(3) $\mathbf{P} = \mathbf{PSPACE}$ なら，任意の時間構成可能関数 $t(n)$ に対し，

$$\mathbf{DTIME}(2^{O(t(n))}) = \mathbf{DSPACE}(2^{O(t(n))}),$$
$$\mathbf{DTIME}(t(n)^{O(1)}) = \mathbf{DSPACE}(t(n)^{O(1)})$$

である．$X(f(n)^{O(g(n))})$ は $\bigcup_c X(f(n)^{cg(n)})$ を表わす．

8 $\mathbf{DLEXP} = \bigcup_{k\geq 0} \mathbf{DTIME}(2^{kn})$ と定義する．**NLEXP** が同様に定義される．$\mathbf{DLEXP} \subsetneq \mathbf{NLEXP}$ かどうかは未解決である．

対角線論法で次のことを示せ．

(a) $\mathbf{P} \subsetneq \mathbf{DLEXP}$ (b) $\mathbf{DLEXP} \subsetneq \mathbf{DEXP}$

9 Rice の定理は $L_{[P]}$ が帰納的言語であるための条件を与えている．$L_{[P]}$ が帰納的可算言語であるための条件については，次のことが知られている[†]．

命題 4.1 $L_{[P]}$ が帰納的可算であるための必要十分条件は，次の (1) ～ (3) が成り立つことである:

(1) L が帰納的可算言語で $L_{[P]}$ の元を部分集合として含むならば，$L \in L_{[P]}$ である．

(2) $L_{[P]}$ の元である任意の無限集合は，$L_{[P]}$ の元であるような有限集合を少なくとも 1 つ部分集合として含む．

(3) $L_{[P]}$ の元である有限集合は可算無限個である．

帰納的可算言語 (TM M) に関する次の性質は，帰納的可算か否か？

(a) $L(M) = \emptyset$ (b) $L(M) \neq \emptyset$ (c) $L(M) = \Sigma^*$ (d) $\varepsilon \in L(M)$

(e) $L(M)$ は帰納的

10 2.4 節 (命題 2.2) で TM と帰納的関数の関係を述べた．すなわち，任意の TM M と任意の正整数 n に対し，

$$M(\overline{x_1}0\overline{x_2}\cdots 0\overline{x_n}) = \overline{y} \Longleftrightarrow \varphi_M^{(n)}(x_1,\ldots,x_n) = y$$

によって n 変数関数 $\varphi_M^{(n)}$ が定義され，このような関数は帰納的部分関数と一致する．

任意の 2 変数の帰納的部分関数 $g(x,y)$ (を定義している TM M) と x とが与えられたとき，y に関する 1 変数関数 $g(x,y)$ を計算する TM M' を (M と x とから) 構成することができる．すなわち，M と x から M' を構成するアルゴリズム (= 停止性 TM) N が存在する．この章で述べたように，TM とそのコードである自然数とを対応させることによって，N は帰納的関数と見ることができる．この事実は，g を $m+n$ 変数関数に一般化して次のように述べなおすことができる (s-m-n **定理**)．これを証明せよ．

命題 4.2 任意の自然数 $\overline{M}, m, n \geq 1$ に対し，

$$\varphi_{\overline{M}}^{(m+n)}(x_1,\ldots,x_m,y_1,\ldots,y_n) = \varphi_{s_n^m(\overline{M},x_1,\ldots,x_m)}^{(n)}(y_1,\ldots,y_n)$$

を満たす $m+1$ 変数の帰納的関数 s_n^m が存在する．

11 (**帰納定理** (recursion theorem)) 任意の帰納的関数 σ に対し，$\varphi_{x_0}^{(1)}(x) = \varphi_{\sigma(x_0)}^{(1)}(x)$ がすべての x に対して成り立つような x_0 が存在することを証明せよ ($\varphi_x^{(1)}$ の意味は問題 10 と同じ)．また，このことを用いて，Rice の定理を証明せよ．

[†] H.G.Rice, On completely recursively enumerable classes and their key arrays, *J. Symbolic Logic* 21, pp.304-341, 1956.

5
還元可能性と完全問題

ある問題を解くことを別の問題を解くことに帰着させることを問題の還元という．還元という手法を使うと，計算不能であることがすでにわかっている問題から別の計算不能な問題を導くこともできるし，計算可能で計算量の上界や下界がすでにわかっている問題から別の問題の計算量の上界や下界を示すこともできる．この章では，最初にいろいろな還元法を導入し，還元手法を使って，計算不能な問題を導く．

還元可能かどうかに従って問題の間に順序を導入することができ，この順序のもとで“もっとも難しい問題”というものを考えることができる．これが“完全問題”である．この章の後半では，さまざまなクラスの完全問題について考察する．それによって，多くの具体的問題について，あるものは計算可能でないことが証明され，あるものは計算可能ではあるが“事実上は計算不能”であると考えてもよいことが証明され，また，あるものは事実上計算不能であろうことが強く予想される．そういった理論的枠組と根拠について考察する．

5.1 還元可能性と計算不能問題

与えられた問題を解くためにその問題を変形して，すでに解く方法が分かっている別の問題に帰着させるという方法は数学においてよく用いられる常套手段の一つである．“還元可能性”と呼ばれる同様の手法は計算量の理論でも用いられ，この分野に実に豊かな成果をもたらした．与えられた問題 A がどのくらい難しいかを判定するために，その問題のサイズを大きく変えないような変形を施して，難しさの程度がすでに分かっている問題 B へと変換する．このように A を B へ変換できる場合，A を解く代わりに B を解けばよいから，A の計算量は (B の計算量)+(変換にかかる計算量) である．もし，変換にかかる

計算量が A の計算量に吸収されてしまうくらい小さいのであれば，A の計算量は B の計算量以下である (なぜなら，A は B に帰着させる以外の方法により，もっと少ない計算量で解けるかもしれないから)．よって，A を B へ変換できることを A $\leq$ B と書く．逆に，A の計算量がすでに分かっているときに A を B へ変換することにより，B の計算量の下界を得ることができる．例えば，A が計算不能なら B も計算不能であることが導かれる．以下では，このような変換方法のいくつかについて考察する．

変換機とは出力テープを持つオフライン DTM M のことであり，入力 x に対する計算が停止したとき，出力テープ上には変換結果として文字列 $M(x)$ が書き出されている (2.4 節参照)．ある多項式 $p(n)$ が存在して M が $p(n)$ 時間限定であるとき M を**多項式時間限定変換機** (polynomial-time bounded transducer) といい，M が $\log n$ 領域限定であるとき**対数領域限定変換機** (logarithmic-space bounded transducer) という．

L, L' を言語とする．条件

$$\text{任意の入力 } x \text{ に対して}, x \in L \Longleftrightarrow M(x) \in L'$$

を満たす停止性変換機 M が存在するとき L は L' に **m-還元可能** (many-one reducible) であるといい，$L \leq_m L'$ と書く．とくに，M が多項式時間限定変換機であるとき L は L' に**多項式時間還元可能** (polynomial-time reducible) であるといい，$L \leq_m^{poly} L'$ と書く．$\leq_m^{poly}$ を **Karp 還元** (Karp reduction) ともいう*．本書では，以後 $\leq_m^{poly}$ を単に $\leq_{poly}$ で表わす．また，M が対数領域限定変換機であるとき L は L' に**対数領域還元可能** (log-space reducible) であるといい，$L \leq_{log} L'$ と書く．とくに，M を明示するために

$$L \leq_{poly} L' \text{ via } M$$

などと書くこともある．

問 5.1 次のことを示せ．
(1) $L \leq_{log} L' \Longrightarrow L \leq_{poly} L' \Longrightarrow L \leq_m L'$.
(2) $\leq$ は $\leq_m$，$\leq_{poly}$，$\leq_{log}$ のどれかとする．$L \leq L' \Longleftrightarrow L^c \leq (L')^c$.

この節では，還元可能性に関する 3 つの重要な補題を示す．最初の一つは次の補題である．

補題 5.1 $L \leq_m L'$ で L が帰納的言語でないならば L' も帰納的言語でない．

証明 $L \leq_m L'$ via M とする．L' が帰納的であるとすると L' を受理する停止性 TM M' が存在する．TM M'' を図 5.1 のように作ると，明らかに M'' は

---

*R.M.Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations*, Plenum Press, pp.85-104, 1972.

停止性 TM であり，

$$x \in L(M'') \Longleftrightarrow M(x) \in L(M') = L' \Longleftrightarrow x \in L$$

であるから $L(M'') = L$ である．これは L が帰納的でないことに反する． □

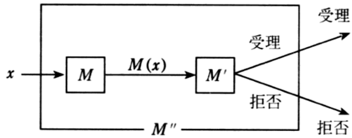


図 5.1 M と M' を合成して M'' を作る

この補題にもとづき，4.4 節で取り上げた問題よりもっと自然な決定不能問題を一つ示してみよう．文法や言語に関する決定不能な問題の多くは，以下に述べる **Post の対応問題** (Post correspondence problem) † から導かれる．

Σ を 2 個以上の記号を含むアルファベットとする．P を $\Sigma^* \times \Sigma^*$ の有限部分集合，すなわち

$$P = \{(x_1, y_1), \ \ldots, \ (x_n, y_n)\}$$

とする．

$$x_{i_1} x_{i_2} \cdots x_{i_k} = y_{i_1} y_{i_2} \cdots y_{i_k}$$

を満たす整数の組 $(i_1, i_2, \ldots, i_k)$, $1 \leq i_j \leq n$, $k \geq 1$, が存在するとき，P は解 $(i_1, i_2, \ldots, i_k)$ を持つという．任意に与えられたこのような P に対して P が解を持つかどうかを決定する問題を Post の対応問題という．この問題を **PCP** で表わす．

例 5.1 (1) $P_1 = \{(01, 011), (11, 1), \ (100, 00)\}$ とする．

$$(x_1, y_1), \ \ (x_2, y_2), (x_3, y_3)$$

$x_1 x_2 x_1 x_3 = 011101100 = y_1 y_2 y_1 y_3$ であるから，$(1, 2, 1, 3)$ は P_1 の解である．

(2) $P_2 = \{(a, b)\}$ は明らかに解を持たない．

(3) $P_3 = \{(ab, ba), (aab, aa)\}$ とする．もし，$(i_1, i_2, \ldots)$ が解だとすると $i_1 = 2$ でなければならない．すると，y 側に欠けている b を補うために，$i_2 = 1$ でなければならない．すると，再び y 側に欠けている b を補うために，$i_3 = 1$ でなければならない．以後，同じことの繰り返しで，P_3 は有限長の解をもたない．
□

† E.L.Post, A variant of a recursively unsolvable problem, *Bull. Amer. Math. Soc.* 52, pp.264-268, 1946.

定理 5.1 Post の対応問題は決定不能である．

証明 補題5.1を使うために，TM に対する帰属問題を Post の対応問題に還元する(すなわち，$L_U \leq_m L_{PCP}$ を示す．ここに，L_{PCP} は Post の対応問題を符号化した言語である．以後，このような形式的な記述はせず，アンダーラインしたような言い方をし，TM-MEMBER $\leq_m$ PCP とも表わす)．

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を $\{0,1\}$ を入力アルファベットとする任意の単テープ TM とし，$x \in \{0,1\}^*$ とする．

$$x \in L(M) \iff P_{M,x} \text{ が解を持つ}$$

が成り立つように Post の対応問題のインスタンス $P_{M,x}$ を構成する方法を以下に示す (形式的には，M, x のコード $\overline{M}\#x$ を入力として，

$$\overline{M}\#x \in L_U \iff N(\overline{M}\#x) = \overline{P_{M,x}} \in L_{PCP}$$

を満たすような $P_{M,x}$ のコード $\overline{P_{M,x}}$ を出力する多項式時間限定変換機 N を構成しなければならないのであるが，以後このように，“符号化” を表に出さない言い方をする．また，停止性変換機のことをアルゴリズムと呼ぶ)．

$Q \cap \Gamma = \emptyset$ とし，M の ID を uqv で表わすことにする ($uv \in \Gamma^*$ は M の現在のテープ内容，q は状態で，ヘッドは v の先頭記号を指している)．$¢, \$, \#$ を $Q \cup \Gamma$ にない新しい記号とし，$\Delta = Q \cup \Gamma \cup \{\#\}$ とおく．さらに，Δ の各記号 X に対し新しい記号 X' を導入し，$\Delta' = \{X' \mid X \in \Delta\}$ とおく．$\xi \in \Delta^*$ に対し，ξ の中の各記号 X を X' で置き換えたものを ξ' で表わすことにする．

$P_{M,x} \subseteq (\Delta \cup \{¢, \$\} \cup \Delta')^* \times (\Delta \cup \{¢, \$\} \cup \Delta')^*$ を，図 5.2のように構成する．$P_{M,x}$ は入力 x に対する M の動作を模倣するように作られている．$(i_1, i_2, \ldots, i_k)$ を $P_{M,x}$ の解とするとき，$(i_1, \ldots, i_j)$, $j \leq k$, を**部分解**と呼び，$x_{i_1} \cdots x_{i_j}$, $y_{i_1} \cdots y_{i_j}$ をそれぞれ部分解の x 部，y 部と呼ぶ．

$$q_0 x \vdash_M \alpha_1 q_1 \beta_1 \vdash_M \cdots \vdash_M \alpha_k q_k \beta_k \quad (\text{各 } q_i \in Q,\ \text{各 } \alpha_i\beta_i \in \Gamma^*)$$

とする (仮に，k は偶数とする)．このとき，$P_{M,x}$ は

$$¢q_0x\#\alpha_1'q_1'\beta_1'\#'\alpha_2q_2\beta_2\#\cdots\#\alpha_{k-1}'q_{k-1}'\beta_{k-1}'$$
$$¢q_0x\#\alpha_1'q_1'\beta_1'\#'\alpha_2q_2\beta_2\#\cdots\#\alpha_{k-1}'q_{k-1}'\beta_{k-1}'\#'\alpha_kq_k\beta_k$$

をそれぞれ x 部，y 部とする部分解を持つ．x 部はつねに y 部より 1 ステップ遅れながら M の動作を模倣する．M の奇数番目のステップに対応する部分語はダッシュ($'$) が付いた語であることに注意する．この部分解はグループ 1~3 だけを使って得られる．もし $q_k \in F$ であれば，このあとさらにグループ 4 を適用してすべてのテープ記号を消去し，最後にグループ 5 を適用して完全な解

$$¢q_0x\#\alpha_1'q_1'\beta_1'\#'\cdots\#'\alpha_kq_k\beta_k\#\cdots\#'q_k\$$$

が得られる．最後のステップでグループ 5 が適用できるためには q_k のすぐ左の # にはダッシュが付いていなければならないので，グループ 5 を適用する直前に，「ダッシュを付けるだけ」という“遊び”のステップが必要な場合もある．以上述べたように，$x \in L(M)$ なら $P_{M,x}$ は解を持つ．

	x_i のリスト	y_i のリスト	条件
グループ 1	¢	$¢q_0x\#$	
グループ 2	X X'	X' X	$X \in Q \cup \Gamma \cup \{\#\}$
グループ 3	ZqX $Z'q'X'$	$p'Z'Y'$ pZY	$\delta(q,X) = (p,Y,-1)$
	qX $q'X'$	$p'Y'$ pY	$\delta(q,X) = (p,Y,0)$
	qX $q'X'$	$Y'p'$ Yp	$\delta(q,X) = (p,Y,1)$
	$Zq\#$ $Z'q'\#'$	$p'Z'Y'\#'$ $pZY\#$	$\delta(q,\flat) = (p,Y,-1)$
	$q\#$ $q'\#'$	$p'Y'\#'$ $pY\#$	$\delta(q,\flat) = (p,Y,0)$
	$q\#$ $q'\#'$	$Y'p'\#'$ $Yp\#$	$\delta(q,\flat) = (p,Y,1)$
グループ 4	XqY $X'q'Y'$ Xq $X'q'$ qY $q'Y'$	q' q q' q q' q	$X, Y \in \Gamma,\ q \in F$
グループ 5	$\#'q\$$	$\$$	$q \in F$

図 5.2　$P_{M,x}$ の構成

逆に，$P_{M,x}$ が解を持つなら $x \in L(M)$ であることを示すのはやや面倒である．最初に，$P_{M,x}$ の解はダッシュが付いた部分語と付かない部分語が交互に現れるようなものしかないことに注意する．また，解はグループ 1 で始まりグループ 5 で終わるものしかないことが容易に分かる．あとは，解の長さに関する帰納法で $x \in L(M)$ であることを証明すればよいが，これは読者への演習問題とする．　□

Post の対応問題の決定不能性を使うことによって，例えば，2 つの文脈自由文法 G_1, G_2 に対して $L(G_1) \cap L(G_2) = \emptyset$ かどうかを決定する問題は計算不能であることが証明できる (演習問題 2 参照).

上述のように，補題 5.1 は決定不能でない問題を示すのには十分有効であるが，変換の計算量に制約がないので計算量の上界を示すのに使うことはできない．そこで，変換に要する計算量を小さめに押さえた次のような還元法を考える．

補題 5.2 (1) $L \leq_{poly} L'$ とする．このとき，次のことが成り立つ:

(a) $L' \in \mathbf{P} \Longrightarrow L \in \mathbf{P}$.　(b) $L' \in \mathbf{NP} \Longrightarrow L \in \mathbf{NP}$.

(c) $L' \in \mathbf{PSPACE} \Longrightarrow L \in \mathbf{PSPACE}$.

(2) $L \leq_{log} L'$ とする．このとき，任意の正整数 k に対して次のことが成り立つ:

(a) $L' \in \mathbf{P} \Longrightarrow L \in \mathbf{P}$,　$L' \in \mathbf{NP} \Longrightarrow L \in \mathbf{NP}$,

$L' \in \mathbf{PSPACE} \Longrightarrow L \in \mathbf{PSPACE}$.

(b) $L' \in \mathbf{DSPACE}(\log^k n) \Longrightarrow L \in \mathbf{DSPACE}(\log^k n)$.

(c) $L' \in \mathbf{NSPACE}(\log^k n) \Longrightarrow L \in \mathbf{NSPACE}(\log^k n)$.

証明 (1) (a) について示す．(b),(c) の証明も全く同様である．$L \leq_{poly} L'$ via M とし，M は $p(n)$ 時間限定であるとする．また，L' を受理する $q(n)$ 時間限定 DTM を M' とする．ただし，$p(n)$, $q(n)$ は多項式である．DTM M'' を図 5.1 のように作ると M'' は L を受理する．M は $p(n)$ 時間限定であるから，入力 x に対する M の出力 $M(x)$ は $|M(x)| \leq p(|x|)$ を満たす．また，M' は $q(n)$ 時間限定であるから，入力 $M(x)$ に対する計算の時間量はたかだか $q(|M(x)|) \leq q(p(|x|))$ である．M による x の変換にかかる時間 $p(|x|)$ を考慮すると，入力 x に対する M'' の時間量は $p(|x|) + q(p(|x|))$ であり，これは $|x|$ の多項式である．よって，$L \in \mathbf{P}$ である．

(2) (a) は問 5.1 の (1) と，この補題の (1) より導かれる．

(b) について示す．(c) は (b) と同様である．(1)(a) の証明と同様に合成 DTM M'' を作る方法は，M が計算した $M(x)$ を保存しておくための領域が必要になり，うまくいかない ($M(x)$ の長さは $O(\log |x|)$ とは限らない)．そこで，M' は M の計算終了を待たずに，M が 1 文字出力するたびにそれを直接使うようにする．M' がヘッドを左に動かさない限りこれでうまくいく．ヘッドを左に動かしたときは，M' は M の計算を最初からやり直してヘッド位置に来るべき文字を求めるようにする．

M'' はカウンタ用テープを 2 本使う．カウンタ 1 には M' の入力ヘッド位置 (すなわち，$M(x)$ の何番目の文字を読んでいるか) を記憶する．M' がヘッドを右 (左) に動かすたびにカウンタ 1 に 1 を加える (1 を減じる)．カウンタ 2 は，M' のヘッド位置にある文字が何であるかを計算するために使う．カウンタ 1 の内容が i のとき，M'' は $M(x)$ の計算を最初からやり直し，M の 1 ステップごとにカウンタ 2 に 1 を加える．カウンタ 1 と 2 の内容が一致したとき，そ

のとき M が出力する文字が求める文字である．M は $\log^k n$ 領域限定であるから，定理 4.1 より M の時間量は $O(c^{\log^k n})$ である (c は定数)．よって，カウンタ 1，2 の長さは $O(\log^k n)$ であり，したがって M の領域量も $O(\log^k n)$ である．線形領域圧縮定理より，$O()$ をはずすことができる．□

問 5.2 $\leq_m$ のもとでは補題 5.2 の (1) が成り立たないことを示せ．また，**NL** $\subsetneq$ **P** と仮定すると (**NL** $\subsetneq$ **P** であるかどうかは現在のところ未解決である)，$\leq_{poly}$ のもとでは (2) (b),(c) が成り立たないことを示せ．

次の補題の証明は補題 5.2 の証明とほとんど同じである．

補題 5.3 $\leq_m$，$\leq_{poly}$，$\leq_{log}$ のどれも反射律，推移律を満たす．

5.2 NP と NP 完全問題

5.2.1 クラス NP

はじめに，**NP** というクラスについて考察しておこう．“非決定性” の意味についてはすでに 3.1 節で考察したが，では，なぜ “多項式” なのであろうか？ コンピュータは，発明されて以来その性能が年々倍々と向上してきたし，そのような性能向上は今後もつづくであろう．しかし，コンピュータの性能がいかに向上しようとも，コンピュータですべての問題が解けるわけではない．計算不能な問題は言わずもがな，たとえ計算可能ではあっても計算量が大きすぎる問題は実質的には計算不能 (intractable) とみなすことができる．例えば，問題のサイズが n のときの計算量が 2^n である問題を計算するためには，たとえ演算速度 (ここでは，計算量の単位である 1 ステップを実行するのにかかる時間のこと) が 1 ピコ秒 (1 秒間に 10^{12} ステップを実行する速さ) のコンピュータ (現時点ではこんなに高速なコンピュータは存在しない) を使っても $n \geq 80$ ならば 1 万年以上かかってしまう．$n = 100$ 程度は普通に遭遇する問題のサイズである．これに対し，計算量が n の多項式，例えば n^6 だとすると，たとえ $n = 100$ であっても実行にかかる時間は，わずか 1 秒である．多項式といっても次数や係数が非常に大きいとやはり実際的な実行時間には収まらない (しかし，多項式時間アルゴリズムを持つ具体的問題の多くが低次の多項式であることも事実である) が，ともかく多項式時間は “実際的” 時間と言ってもよいと思われる．

一方，歴史的に TM をはじめとして実にさまざまの計算モデルが提案されたが，多項式時間ということに関してはどのモデルも同値であることが証明されている．すなわち，あるモデルのもとで多項式時間で計算できる問題は他のモ

デルにおいても多項式時間で計算できる (ただし，多項式の次数は異なるかもしれない)．このように，多項式時間で計算可能な問題のクラスはモデルに依存しない安定したクラスである．多項式のもう一つの魅力は，合成で閉じている等，理論を展開する上で必須の数学的に望ましい性質を持っていることである．例えば，多項式時間還元可能性による“完全問題”の定義はこの性質に負う．

次の定理は，**NP** とはどういうクラスなのかについてわずかながら洞察を与えてくれる．

述語 (predicate) とは論理値 (**true**/**false**) を値とする関数のことである．2 変数の述語を計算する多項式時間限定アルゴリズム (**true**/**false** を出力する変換機) M と定数 k が存在して

$$L = \{x \mid M(x, y) = \mathbf{true} \text{ かつ } |y| = O(|x|^k) \text{ となる } y \text{ が存在する}\}$$

であるとき，M は L を**多項式時間で検証** (verify in polynomial time) するという．$M(x, y) = \mathbf{true}$ なる y を x の**確認証** (certificate) という．

定理 5.2 $L \in \mathbf{NP}$ であることと，L を多項式時間で検証するアルゴリズムが存在することとは同値である．

証明 $L \in \mathbf{NP}$ とすると，L を受理する n^k 時間限定 NTM N と定数 k が存在する．入力 $x \in L$ に対する M の受理計算 (ID の列) を文字列とみなしたものを y とすると 1 つの ID の長さはたかだか $O(|x|^k)$ であるから，明らかに $|y| = O(|x|^{2k})$ である．x, y を入力したとき，y が x の受理計算だったら **true** を出力し，そうでなかったら **false** を出力する多項式時間限定変換機 M を容易に構成することができる．よって，L は多項式時間で検証される．

逆に，L を多項式時間で検証するアルゴリズム M と定数 k が存在したとする．次のような NTM M' を考える．入力 x が与えられたら，M' は $|y| \leq |x|^k$ なる $|y|$ を非決定的に生成し，x, y を入力として M を模倣する．$M(x, y) = \mathbf{true}$ だったら M' は x を受理し，$M(x, y) = \mathbf{false}$ だったら拒否する．明らかに，M' は L を受理し，かつ多項式時間限定である．よって，$L \in \mathbf{NP}$． □

この定理の意味を考えてみよう．問題 L (例えば，ハミルトン閉路問題) のインスタンス x (すなわち，グラフ G) が与えられたとき，もし $x \in L$ である (G がハミルトン閉路を持つ) ならば，あまり大きくない ($|x|$ の多項式) サイズの確認証 y (例えば，G の辺の列) をうまく選べば $x \in L$ であることを多項式時間で検証することができる．このようなことが可能なのが **NP** に属す問題なのである，と定理は主張している．比喩的に言うと，**NP** に属す問題とは，問題の具体例が与えられたとき，直感を働かせてうまく確認証を見つけることができれば多項式時間で解くことができるような問題のことである．この節で後

述するように，現実世界で遭遇する問題の多くがこのような性質を持っている．したがって，そのような問題 (**NP** に属す問題) が実際的に計算可能であるかどうかは重要な関心事である (上述したように，**P** は実際的に計算可能 (feasible) な問題のクラスと考えることができる)．しかし，**NP** に属すことは容易に分かるのに **P** に属すか否かが分からない問題がたくさんある．いったい "非決定性" にはどれだけのパワーが秘められているのであろうか？ いくつかの特殊な計算量のクラスを除いて (例えば,

$$\mathbf{DSPACE}(1) = \mathbf{NSPACE}(1),\ \mathbf{DTIME}(n) \subsetneq \mathbf{NTIME}(n),$$
$$\bigcup_i \mathbf{DSPACE}(n^i) = \bigcup_i \mathbf{NSPACE}(n^i)$$

など)，多くの計算量のクラスについて決定性と非決定性に能力差があるのかどうか未解決のままである．多項式時間に関してもそうである．$\mathbf{P} = \mathbf{NP}$ なのか? それとも $\mathbf{P} \subsetneq \mathbf{NP}$ なのか? $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ **問題**と呼ばれるこの有名な未解決問題の解決にとって，以下で述べる **NP** 完全問題は重要な役割を持つ．もし，ある **NP** 完全問題 (どの **NP** 完全問題でもよい) を解く決定性多項式時間アルゴリズムが見つかれば $\mathbf{P} = \mathbf{NP}$ であることが導かれ，その結果，何千という問題が一挙に多項式時間で解けてしまうことが知られている．

きちんと定義をしよう．$\mathcal{C}$ を問題 (= 言語) のクラスとする．言語 L_0 が多項式時間 (対数領域) 還元性に関して $\mathcal{C}$ **困難** ($\mathcal{C}$-hard) であるとは，$\mathcal{C}$ の任意の元 L が L_0 に多項式時間 (対数領域) 還元可能なことをいう．$\mathbf{NP} \subseteq \mathcal{C}$ の場合，L_0 が 多項式時間還元性に関して $\mathcal{C}$ 困難かつ $L_0 \in \mathcal{C}$ であるとき，L_0 は $\mathcal{C}$ **完全** ($\mathcal{C}$-complete) であるという．$\mathbf{NL} \subseteq \mathcal{C} \subseteq \mathbf{P}$ の場合，L_0 が 対数領域還元性に関して $\mathcal{C}$ 困難かつ $L_0 \in \mathcal{C}$ であるとき，L_0 は $\mathcal{C}$ 完全であるという．荒っぽく言うと，$\mathcal{C}$ 困難問題とは $\mathcal{C}$ のどの問題よりも難しい (計算量が大きい) 問題のことであり ($\mathcal{C}$ に属すとは限らない)，$\mathcal{C}$ 完全問題とは $\mathcal{C}$ の中で最も難しい問題のことである．例えば，この章の表題である **NP 完全問題**とは，$L_0 \in \mathbf{NP}$ かつ任意の $L \in \mathbf{NP}$ に対して $L \leq_{poly} L_0$ であるような L_0 のことである．

完全性の定義をする際，**NP** を含むクラスと **P** に含まれるクラスに対して還元可能性を使い分けている理由は，$\mathcal{C}$ に属する問題を分類するためには，還元に要する計算量が $\mathcal{C}$ の問題自体が持つ計算量よりも小さくなければならないからである (問 5.2 参照)．しかし，**NP** 以上のクラスに対して完全であることが証明されている問題の多くは (すべてというわけではない)，対数領域還元性に関しても完全である．

問 5.3 A が $\mathcal{C}$ 完全で $A \leq_{log} B$, $B \in \mathcal{C}$ ならば B も $\mathcal{C}$ 完全であることを示せ．

$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ 問題等，主要クラス間の未解決な関係を明らかにするためにも，完全問題は一つのキーになり得る:

定理 5.3 (a) L を **NP** 完全問題または co-**NP** 完全問題とすると，$L \in \mathbf{P} \Longleftrightarrow \mathbf{P} = \mathbf{NP}$.

(b) L を co-**NP** 完全問題とすると，$L \in \mathbf{NP} \Longleftrightarrow \mathbf{NP} = \text{co-}\mathbf{NP}$.

(c) L を **PSPACE** 完全問題とすると，$L \in \mathbf{P} \Longleftrightarrow \mathbf{P} = \mathbf{PSPACE}$.

(d) L を **P** 完全問題とすると，$L \in \mathbf{NL}$ $(L \in \mathbf{DL}) \Longleftrightarrow \mathbf{NL} = \mathbf{P}$ $(\mathbf{DL} = \mathbf{P})$.

(e) L を **NL** 完全問題とすると，$L \in \mathbf{DL} \Longleftrightarrow \mathbf{DL} = \mathbf{NL}$.

証明 (a) についてだけ証明する．他も同様である．

L を **NP** 完全問題とすると任意の $L' \in \mathbf{NP}$ に対して $L' \leq_{poly} L$ であるから，補題 5.2 (1) より，$L \in \mathbf{P}$ ならば $L' \in \mathbf{P}$ である．よって，$\mathbf{NP} \subseteq \mathbf{P}$. 同様に，$L \in \mathbf{P}$ が co-**NP** 完全問題であるとすると co-$\mathbf{NP} \subseteq \mathbf{P}$ が導かれるが，**P** が補集合で閉じているので $\mathbf{NP} \subseteq \text{co-}\mathbf{P} = \mathbf{P}$. よって，いずれの場合も $\mathbf{P} = \mathbf{NP}$ である．

逆に，$\mathbf{P} = \mathbf{NP}$ ならば $L \in \mathbf{P}$ であることは明らかである． □

還元可能性とか，完全問題といった概念は，もともと帰納的関数の理論において導入されて豊かな成果をもたらしたものであるが[‡]，S.A.Cook はこれを巧みに計算量理論に取り込んで **NP** 完全問題の存在を初めて示した[§]．それにつづいて，R.M.Karp が数多くの組合わせ論的問題が **NP** 完全になることを示してその重要性を指摘した[¶]．これを契機として各種のクラスに対する完全問題の研究が活発に行なわれ，その後の計算量理論の発展の大きな原動力となった．

Cook が最初に示した，クラス **NP** に対する完全問題は**ブール式の充足可能性判定問題** (**SAT**: satisfiability problem for Boolean expressions) である:

入力: ブール式 E
出力: E が充足可能か否か

定理 5.4 (Cook の定理) SAT は **NP** 完全である．

証明 ブール式 E が与えられたとき，その中の各ブール変数に対し，E を充足させるような 0 または 1 を非決定的に推測し，実際そのときの E の値を計算して推測を検証することは $O(n^2)$ 時間でできるから，SAT $\in$ **NP** である (ブール式 E をアルファベット $\{x, 0, 1, \neg, \wedge, \vee, (,)\}$ 上の語 $\overline{E}$ として適当

[‡]例えば，H.Rogers Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967 を参照せよ．

[§]S.A.Cook, The complexity of theorem-proving procedures, *Proc. 3rd Annual ACM Symp. on the Theory of Computing*, pp.151-158, 1971. この論文では Turing 還元 (演習問題 3 参照) が用いられた．

[¶]R.M.Karp, Reducibilities among combinatorial problems, in *Complexity of Computer Computations* edited by J.W.Thatcher and R.E.Miller, Plenum Press, pp.85-103, 1972. この論文で多項式時間還元が導入された．

に符号化し，$L_{SAT} = \{\overline{E} \mid E$ は充足可能なブール式$\}$ とする．SAT $\in$ **NP** は $L_{SAT} \in$ **NP** の略記であり，n は入力 $\overline{E}$ の長さである．同様に，$L \leq_{poly}$ SAT は $L \leq_{poly} L_{SAT}$ の略記である)．

SAT が **NP** 困難であることを示そう．すなわち，$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を多項式時間限定の任意の NTM とするとき，$L(M) \leq_{poly} SAT$ であることを示す．M は $p(n) = n^k$ 時間限定であるとする．以下でブール式を構成する際，$p(n)$ の 2 進数表現を使う．$p(n)$ の 2 進数表現は対数領域構成可能である (3.2.3 節参照)．定理 2.1 (2.2 節)，系 3.1 (3.2 節) より，M は片方向無限テープを持つ単テープ NTM であると仮定してもよい．入力 x に対し

$$x \in L(M) \Longleftrightarrow E_x \text{ が充足可能}$$

を満たすブール式 E_x を出力する多項式時間限定の変換機が存在することを示せばよい (すなわち，この変換機を N とすると，$x \in L(M) \Longleftrightarrow E_x = N(x) \in L_{SAT}$ が成り立つ)．

$x = a_1 a_2 \cdots a_n$ $(a_i \in \Sigma)$ を M への入力とする．M の ID は，定理 5.1 の証明と同様に，uqv なる形の文字列であるとする ($Q \cap \Gamma = \emptyset$，$q \in Q$ は状態記号，$uv \in \Gamma^*$ はテープの内容，入力ヘッドは v の先頭文字の上にある)．すべての ID は長さがちょうど $p(n)+1$ であるとする (これより短いものは $\flat$ を補って考える)．M は，一旦受理状態に入ったらそのまま同じ状態を続けるものと仮定してもよい．このとき，$x \in L(M)$ である必要十分条件は，次の条件を満たす ID の列 $\alpha_0, \alpha_1, \ldots, \alpha_{p(n)}$ が存在することである:

(1) α_0 は入力 x に対する M の初期 ID である．すなわち，$\alpha_0 = q_0 x \flat^{p(n)-n}$.

(2) 任意の i $(1 \leq i \leq p(n))$ に対して $\alpha_{i-1} \vdash_M \alpha_i$.

(3) $\alpha_{p(n)}$ は受理 ID である．すなわち，$\alpha_{p(n)} = uqv$ となる $q \in F$, $uv \in \Gamma^*$ が存在する．

さて，(1)〜(3) をブール式で表わすために，ブール変数

$$X_{ija} \quad (0 \leq i \leq p(n),\ 1 \leq j \leq p(n)+1,\ a \in \Gamma)$$

を考える．X_{ija} によって，α_i の左から j 番目の記号が a であることを表わしたい (すなわち，$X_{ija} = 1 \Longleftrightarrow \alpha_i$ の左から j 番目の記号が a)．このためには，条件

(4) 任意の i, j $(0 \leq i \leq p(n),\ 1 \leq j \leq p(n)+1)$ に対して，$X_{ija} = 1$ となるような $a \in Q \cup \Gamma$ がちょうど 1 つだけ存在する

が満たされていなければならない．

(1)〜(4) はそれぞれ以下のようなブール式で表わすことができる．

$$E_{(1)} = X_{01q_0} \wedge \left(\bigwedge_{2\le i\le n+1} X_{0ia_i}\right) \wedge \left(\bigwedge_{n+2\le i\le p(n)+1} X_{0i\flat}\right).$$

ここで，$\bigwedge_{P(i)}\psi(i)$ は $P(i)$ を満たすようなすべての i に関して $\psi(i)$ の論理積をとることを表わし，「$P(i)$ を満たすすべての i に対して $\psi(i)$ が成り立つ」ことを表わしていることに注意する．同様に，$\bigvee_{P(i)}\psi(i)$ は $P(i)$ を満たすすべての i に関して $\psi(i)$ の論理和をとることを表わし，「$P(i)$ を満たす i の中に $\psi(i)$ を成り立たせるものが存在する」ことを表わしている．

$$E_{(2)} = \bigwedge_{1\le i\le p(n)} (L(i) \vee S(i) \vee R(i)).$$

ここに，$L(i), S(i), R(i)$ は，テープヘッドをそれぞれ $-1, 0, 1$ こま右に動かすような動作によって遷移 $\alpha_{i-1} \vdash_M \alpha_i$ が行なわれることを表わすブール式である．例えば，

$$R(i) = \bigvee_{\substack{(p,b,1)\in\delta(q,a)\\ 1\le j\le p(n)}} \left(X_{i-1,\,j,\,q} \wedge X_{i-1,\,j+1,\,a} \wedge X_{ijb} \wedge X_{i,\,j+1,\,p} \wedge EQ(j)\right)$$

である．ただし，$EQ(j)$ は，$k \ne j$ なるすべての k $(1 \le k \le p(n)+1)$ に対して α_{i-1} の k 番目の記号と α_i の k 番目の記号とが等しいことを表わすブール式

$$EQ(j) = \bigwedge_{\substack{1\le k\le p(n)+1\\ k\ne j}} \bigvee_{a\in\Gamma} \left(X_{i-1,\,k,\,a} \wedge X_{ika}\right)$$

である．$L(i), S(i)$ も $R(i)$ と同様に表わすことができる．

$$\begin{aligned} E_{(3)} &= \bigvee_{1\le j\le p(n)+1} \bigvee_{q\in F} X_{p(n),\,j,\,q}. \\ E_{(4)} &= \bigwedge_{\substack{0\le i\le p(n)\\ 1\le j\le p(n)+1}} \left[\bigvee_{a\in Q\cup\Gamma} X_{ija} \wedge \neg\left(\bigvee_{\substack{a,b\in\Gamma\\ a\ne b}} (X_{ija} \wedge X_{ijb})\right)\right]. \end{aligned}$$

求めるブール式 E_x は

$$E_x = E_{(1)} \wedge E_{(2)} \wedge E_{(3)} \wedge E_{(4)}$$

である．$|E_x| = O((p(n)\log_2 p(n))^2)$ であること，および，入力 x に対し E_x (のコード $\overline{E_x}$) を出力する，$O(\log(p(n)\log_2 p(n))^2) = O(\log n)$ 領域限定 (したがって多項式時間限定) の変換機が構成できることは明らかであろう．$\log_2 p(n)$ が必要な理由は，E_x 内のすべてのブール変数 (の添字) は 2 進数を使って符号化する必要があることによる．

最後に，$x \in L(M) \iff E_x$ が充足可能，であることを示そう．$x \in L(M)$ とすると，(1)〜(3) を満たす $\alpha_0, \alpha_1, \ldots, \alpha_{p(n)}$ が存在する．α_i の左から j 番目の記号が a であるなら $X_{ija} = 1$，そうでないなら $X_{ija} = 0$ とすれば，E_x の構成の仕方より，このような変数の値のもとで E_x の値は 1 である．ゆえに，E_x は充足可能である．

逆に，E_x が充足可能であるとすると，E_x の値を 1 とするように各変数に 0/1 を割り当てることができ，$E_{(1)} \sim E_{(4)}$ それぞれの値は 1 である．$E_{(4)} = 1$ であることより，各 i, j $(0 \le i \le p(n),\ 1 \le j \le p(n)+1)$ に対して $X_{ija} = 1$ となる $a \in Q \cup \Gamma$ が一意的に存在する．よって，

$$\alpha_i \text{ の左から } j \text{ 番目の記号} = a \iff X_{ija} = 1$$

として，長さ $p(n)+1$ の文字列 $\alpha_0, \alpha_1, \ldots, \alpha_{p(n)}$ を定義すれば (1)~(3) が成り立つ．すなわち，入力 x に対する M の受理計算が存在する．ゆえに，$x \in L(M)$ である．　□

問 5.4 定理 5.4 の証明に関して，次の各問に答えよ．

(1) 与えられた NTM が $p(n)$ 時間限定であるかどうかを決定することは計算不能である (証明せよ)．それにもかかわらず，定理 5.4 の証明で $p(n)$ を具体的に与えているのはなぜか？

(2) 条件 (4) は，M が非決定性であるがゆえに必要とされるものである．その理由を説明せよ．

どんなブール式も「リテラルの論理和」の論理積 (CNF: 乗法標準形) で表わせることをすでに定理 1.4 で示した (その証明は，一般のブール式をそれと同値な CNF に変換するアルゴリズムになっているが，このアルゴリズムは指数関数時間アルゴリズムである)．CNF において，リテラルの論理和のことを節(せつ) (clause) と呼ぶ．CNF を構成する各節がちょうど k 個のリテラルの論理和となっているとき，この CNF を ***k*-CNF** という．3-CNF のブール式だけに限定すると，充足可能性は制限なしの場合よりも効率良く判定できるであろうか？答は no である．

定理 5.5 3SAT (与えられた 3-CNF のブール式が充足可能であるか否かを決定する問題) は **NP** 完全である．

証明 3SAT は SAT の特別の場合であるから 3SAT $\in$ **NP** は明らかである．3SAT が **NP** 困難であることを SAT $\le_{poly}$ 3SAT によって示す．

ステップ 1　ブール式 ϕ が与えられたら，その構文木を図 5.3 のように作る．ϕ の**構文木** (parse tree) とは，ϕ の構造を 2 分木で表わしたものである (形式的な定義は省略するが，説明は不要であろう)．$\vee$ と $\wedge$ は結合律を満たすので演算順を規定する括弧対が省略されていることもある (図 5.3 の ϕ では $x_1 \vee x_2 \vee \ldots$ の部分) が，その場合には結合順を適当に決めてやる．

構文木はブール式の計算順を表わす木となっている．各頂点での出力を表わすブール変数 y_i を導入して，ϕ を次のように ϕ' に書き換える (以下，簡単のために例だけで説明する)：

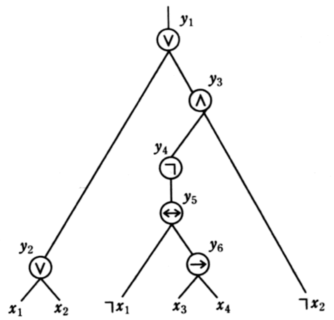


図 5.3　$\phi = (x_1 \vee x_2) \vee \neg(\neg x_1 \leftrightarrow (x_3 \rightarrow x_4)) \wedge \neg x_2$ の構文木

$$
\begin{aligned}
\phi' = y_1 & \wedge (y_1 \leftrightarrow (y_2 \vee y_3)) \\
& \wedge (y_2 \leftrightarrow (x_1 \vee x_2)) \\
& \wedge (y_3 \leftrightarrow (y_4 \wedge \neg x_2)) \\
& \wedge (y_4 \leftrightarrow \neg y_5) \\
& \wedge (y_5 \leftrightarrow (\neg x_1 \leftrightarrow y_6)) \\
& \wedge (y_6 \leftrightarrow (x_3 \rightarrow x_4)).
\end{aligned}
$$

このとき，ϕ が充足可能である必要十分条件は ϕ' が充足可能なことである (問：証明せよ)．一般に ϕ から ϕ' を生成する多項式時間限定のアルゴリズムが存在することは明らかであろう．

ステップ 2　ϕ' の中の $\leftrightarrow$ を含む部分論理式 (y_i の個数と同数だけある．y_i に対応するものを ϕ'_i とする) のそれぞれを次のように書き換える．例えば，$\phi'_1 = y_1 \leftrightarrow (y_2 \vee y_3)$ を考えよう．ϕ'_1 の真理表を作り，定理 1.4 に従って ϕ'_1 の CNF ϕ''_1 を作る:

y_1	y_2	y_3	$y_1 \leftrightarrow (y_2 \vee y_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$
\begin{aligned}
\phi''_1 = & \quad (y_1 \vee y_2 \vee \neg y_3) \\
& \wedge (y_1 \vee \neg y_2 \vee y_3) \\
& \wedge (y_1 \vee \neg y_2 \vee \neg y_3) \\
& \wedge (\neg y_1 \vee y_2 \vee y_3)
\end{aligned}
$$

ϕ' の中の各 ϕ'_i を ϕ''_i で置き換えたものを ϕ'' とすると，$\phi' = \phi''$ (ϕ' と ϕ'' は恒等的に等しい) であり，ϕ'' はたかだか 3 個のリテラルの論理和からなる CNF である．各 ϕ'_i はたかだか 3 個の変数しか含まないので，各真理表のサイズは $O(1)$ であり，各 ϕ''_i は $O(1)$ 時間で，したがって ϕ'' は y_i の個数に関する多項式時間で作ることができる．

ステップ 3　ϕ'' を 3-CNF へ変換する．ϕ'' の各節 φ''_i を次のように φ'''_i に変換する．

(i) φ''_i がちょうど 3 個のリテラルからなるとき，$\varphi'''_i = \varphi''_i$.

(ii) φ''_i が 2 個のリテラルからなるとき，$\varphi'''_i = (\varphi''_i \vee p) \wedge (\varphi''_i \vee \neg p)$ とする．ここに，p は新しいブール変数．

(iii) φ''_i が 1 個のリテラルだけからなるとき，$\varphi'''_i = (\varphi''_i \vee p \vee q) \wedge (\varphi''_i \vee p \vee \neg q) \wedge (\varphi''_i \vee \neg p \vee q) \wedge (\varphi''_i \vee \neg p \vee \neg q)$ とする．ここに，p, q は新しいブール変数．

(i)〜(iii) いずれにおいても，p, q の値とは無関係に $\varphi''_i = \varphi'''_i$ が成り立つ．ゆえに，ϕ'' の中の各 φ''_i を φ'''_i で置き換えたものを ϕ''' とすると，$\phi''' = \phi''$ である．ϕ''' は ϕ'' の長さの線形時間で作ることができる．

ϕ''' は 3-CNF であり，ϕ が充足可能である必要十分条件は ϕ''' が充足可能なことである．また，以上 3 つのステップを通した総時間量は入力 ϕ の長さの多項式である．　□

系 5.1 どんなブール式も 3-CNF で表わすことができる．

定理 5.5 より，$k \geq 3$ なら kSAT は **NP** 完全である (問: 証明せよ) が，2SAT は **NL** 完全であることを 5.3 節で示す．一方，1SAT $\in$ **DL** である．

問 5.5 (1) DNF (加法標準形) のブール式が充足可能か否かを判定する問題は **P** に属すことを示せ．

(2) 次の証明はなぜ正しくないのか？「ブール式が与えられたら，定理 1.4 の方法で DNF を求めれば，(1) によって充足可能性を多項式時間で判定できる．よって，SAT は **P** に属す.」

(3) 1SAT $\in$ **DL** であることを証明せよ．

与えられたグラフがハミルトン閉路を持つか否かを判定する問題を**ハミルトン閉路問題** (HAMILTON) という (2.3 節の例 2.5 (2)).

定理 5.6 HAMILTON は **NP** 完全である．

証明 HAMILTON $\in$ **NP** であることは容易に示すことができる (例 3.1 (2)). 以下で，3SAT が HAMILTON に多項式時間還元可能であることを示すことによって，HAMILTON が **NP** 困難であることを示す．

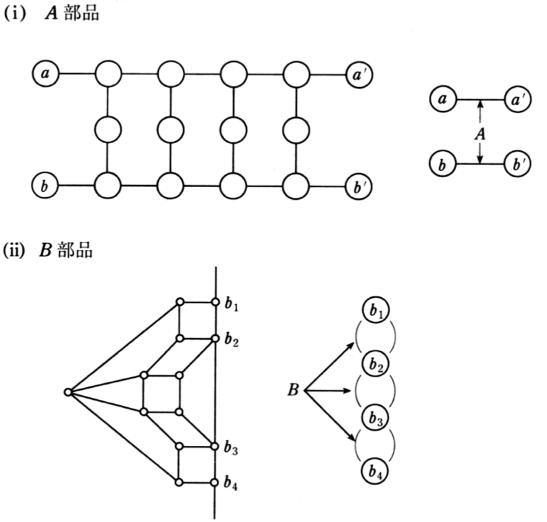


図 5.4　2 つの部分グラフ (左) とその省略形 (右)

ϕ を 3-CNF のブール式とし，ϕ の変数を $x_1,\ldots,x_n$，ϕ の節を $C_1,\ldots,C_k$ とする．ϕ が与えられたとき

$$\phi \text{ が充足可能} \iff G=(V,E) \text{ はハミルトン閉路を持つ}$$

を満たすグラフ G を多項式時間で構成するアルゴリズムを示す．G を構成するために，図 5.4 に示した 2 つのグラフを部品として使う．

図 5.4 (i) 左図のグラフ (A **部品**と呼ぼう) が G の中で ⓐ，ⓑ，ⓐ′，ⓑ′ においてだけ他の部分と接続しているとすると，G がハミルトン閉路を持つとしたときその閉路は，図 5.5 に示したような 2 つの道のどちらかを通るしかない．そこで，A 部品を，図 5.4 (i) 右図のように略記することにする．これは，G のハミルトン閉路は aa', bb' のどちらか一方だけを必ず通らなければならないことを表わす．

図 5.4 (ii) 左図のグラフ (B **部品**と呼ぼう) が G の中で (b_1) ～ (b_4) においてのみ他の部分と接続しているとすると，G のハミルトン閉路は辺 b_1b_2，b_2b_3，b_3b_4 のすべてを通ることはない (もしそうだとすると，B 部品のこれらの辺以外の部分は通らないことになってしまい，それはハミルトン閉路ではない) が，それ以外 (すなわち，b_1b_2，b_2b_3，b_3b_4 のうちの少なくとも 1 つを通らない) 場合

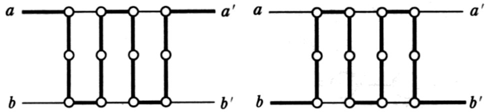


図 5.5 部分グラフ (i) を通るハミルトン道

にはいずれも，B 部品のすべての頂点をちょうど 1 回ずつ通るようにできる．そこで，この部分グラフを，図 5.4 (ii) 右図のように略記することにする．これは，G のハミルトン閉路は B からの矢印が指している辺のうちの少なくとも 1 つを通らなければならないことを表わす．

G を以下のように構成する．各節 C_i ごとに，図 5.6 左図のような部分グラフを作り，b_{i4} と $b_{i+1,1}$ を辺で結ぶ $(1 \leq i \leq k-1)$．また，各変数 x_m ごとに，図 5.6 右図のような部分グラフを作り，x''_m と x'_{m+1} を辺で結ぶ $(1 \leq m \leq n-1)$．また，b_{11} と x'_1，b_{k4} と x''_n をそれぞれ辺で結ぶ．

最後に，変数と節がどのように関係しているかを表わすために，図 5.6 右図の 2 つの A からの矢印を次のように結ぶ:

(a) C_i の中の j 番目のリテラルが x_m なら，$\overline{e_{ij}}$ と e_m を同じ A からの矢印で結ぶ (すなわち，$\overline{e_{ij}} \leftarrow \mathrm{A} \rightarrow e_m$ とする)．

(b) C_i の中の j 番目のリテラルが $\neg x_m$ なら，$\overline{e_{ij}}$ と $\overline{e_m}$ を同じ A からの矢印で結ぶ (すなわち，$\overline{e_{ij}} \leftarrow \mathrm{A} \rightarrow \overline{e_m}$ とする)．

ただし，例えば，図 5.7 左図の実際の構成は同右図のようになる．以上で G の構成が終わった (図 5.8 に例を示した)．ϕ から G を作ることは多項式時間でできる．

$$\phi \text{ が充足可能} \iff G \text{ はハミルトン閉路を持つ}$$

を証明すればよい．

$(\Longrightarrow)$ ϕ を充足させるような $x_1, \ldots, x_n$ への値の割り付けに対して，次のように辺を選ぶ:

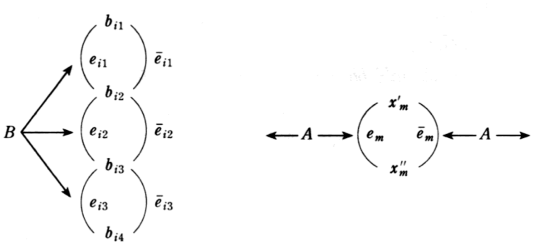


図 5.6 リテラル C_i 対する B 部品 (左) と，変数 x_m に対する A 部品 (右)

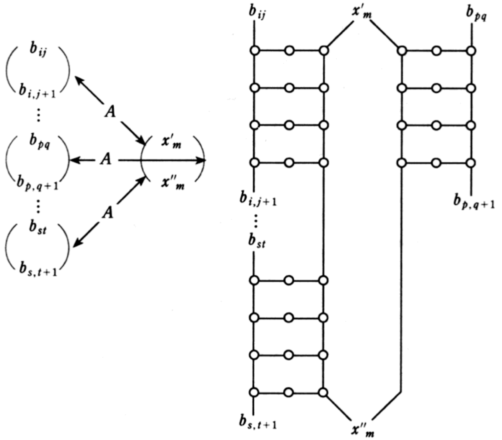


図 5.7　略記法 (左) と実際の構成 (右)

(1) $x_m = 1$ なら e_m をたどる．

(2) $x_m = 0$ なら $\overline{e_m}$ をたどる．

(3) C_i の j 番目のリテラルの値が 0 なら $\overline{e_{ij}}$ をたどり，1 なら $\overline{e_{ij}}$ をたどらない (このことを「e_{ij} をたどる」ということにする．実際は，e_{i1}，e_{i2}，e_{i3} のうちのどれとどれをたどるかによって異なるが，いずれの場合にも B 部品のすべての頂点をちょうど 1 回ずつたどることができることをすでに述べた)．

C_i の第 j リテラルが x_m で $x_m = 1$ の場合，(a) と (1) より，e_{ij} をたどり，$x_m = 0$ の場合，(a) と (2) より，$\overline{e_{ij}}$ をたどれば矢印 A についての条件 (同じ A からの矢印が指す 2 辺のどちらか一方だけを必ず通ること) が満たされる．同様に，第 j リテラルが $\neg x_m$ で $x_m = 1$ の場合，(b) と (1) より，$\overline{e_{ij}}$ をたどり，$x_m = 0$ の場合，(b) と (2) より，e_{ij} をたどる．このようにすると，第 j リテラルの値が 1 なら e_{ij} をたどり，0 なら $\overline{e_{ij}}$ をたどることになる．ϕ の値が 1 のときにはどの節も少なくとも 1 つのリテラルの値が 1 であるから，矢印 B についての条件 (B からの矢印が指している辺のうちの少なくとも 1 つを通ること) が満たされる．条件 A, B が満たされれば，すでに述べたように G はハミルトン閉路を持つ．

($\Longleftarrow$) 節 C_i に対応する B 部品について考える．G のハミルトン閉路は 3 辺

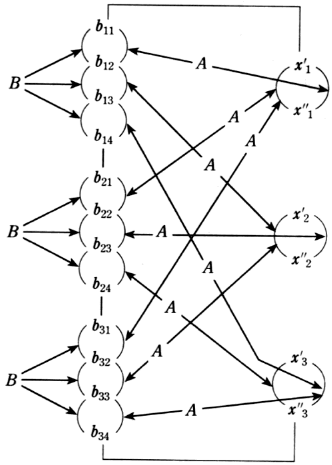


図 5.8 $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$ に対するグラフ G

$\overline{e_{i1}}$, $\overline{e_{i2}}$, $\overline{e_{i3}}$ のすべてをたどることはない．そのたどられない辺に対応するリテラルの値が 1 になるように変数に 0/1 を割り当てると，節 C_i の値は 1 になる．ハミルトン閉路は A についての条件も満たしていなければならないので，G の構成の仕方 (異なる節に属する同じリテラルは同じ e_m (または $\overline{e_m}$) と A 矢印で結ばれていること) より，このような変数への値の割り付けは異なる C_i によって矛盾することはない．よって，すべての C_i を 1 とするような変数への値の割り付けが存在するので，ϕ は充足可能である． □

ハミルトン閉路問題は，頂点の最大次数が 3 以下の平面グラフ (どの 2 辺も交差しないように平面上に描くことができるグラフ) に制限しても，あるいは 4 重連結グラフ (どの 3 頂点とそれに接続する辺を除いても連結のままであるようなグラフ) に制限しても **NP** 完全であることが知られている．最大次数を 2 以下に制限すると **P** に属す (問: 証明せよ)．

$\mathbf{Z}$ を整数すべての集合，$k \in \mathbf{Z}$ とし，$G = (V, E)$ を辺に重み $w : E \to \mathbf{Z}$ が付いた完全グラフとする．G, w, k が与えられたとき，辺の重みの和が k 以下であるようなハミルトン閉路が G に存在するか否かを判定する問題を**巡回セールスマン問題** (TSP: traveling salesman problem) という．これは，あるセールスマンが自分の受け持ち区域内の $|V|$ 個の都市のすべてをできるだけ効率良く巡回するにはどうしたらよいか，という最適化問題を決定問題として定式化し

たものである．辺 ij の重み $w(i,j)$ は都市 i と j の間を移動するのに要するコストを表わす．

系 5.2 TSP は **NP** 完全である．

証明　TSP $\in$ **NP** であることは容易に分かる．HAMILTON $\leq_{poly}$ TSP を示す．グラフ $G=(V,E)$ が与えられたとき，グラフ $G'=(V,E')$ と重み関数 $w: E' \to \mathbf{Z}$ を次のように定義する:

$$E' = V \times V, \quad w(i,j) = \begin{cases} 0 & ij \in E\text{のとき} \\ 1 & ij \notin E\text{のとき}. \end{cases}$$

明らかに，G' と w は多項式時間で求めることができる．G がハミルトン閉路を持つ $\Longleftrightarrow$ G' が辺の重みの和が 0 以下のハミルトン閉路を持つ，が成り立つことが容易に確かめられる．　□

その他の NP 完全問題

すでに述べたように，**NP** 完全であることがわかっている問題のどの 1 つでもよいから多項式時間で解けることを示すことができれば，すべての **NP** 問題が多項式時間で解けることになる．これまでに **NP** 完全問題の例をいくつか示したが，実は，ごく日常的に出会う問題の中で **NP** 完全であることが知られているものが何千とある．そういった例をいくつか列挙しておこう[‖]．これらの問題を解くためのアルゴリズムとしては，すべての場合をしらみつぶしに調べるという素朴な方法 (指数関数的時間がかかる) を本質的に打ち破るような効率良い方法はまったく知られていない．

(1) **ナップザック問題** (knapsack)　大きさがそれぞれ $w_1,\ldots,w_n$，価値がそれぞれ $v_1,\ldots,v_n$ である n 個の品物 (それぞれ 1 つずつ) と，容量 W のナップザックが与えられたとき (w_i, v_i, W はすべて整数)，大きさの和が W 以下で価値の和が最大になるように，ナップザックに詰める品物を選べ．

形式的には，さらに整数 V を与えて，$\sum_{i\in I} w_i \leq W$ かつ $\sum_{i\in I} v_i \geq V$ を満たすような $I \subseteq \{1,\ldots,n\}$ が存在するか否かを判定する "決定問題" として考える必要がある．以下の問題についても同様である．

(2) **箱詰め問題** (bin packing)　容量が C の箱がいくつもある．n 個の品物 $a_1,\ldots,a_n$ と各品物 a_i の大きさ $S(a_i)$ が与えられたとき，これらの品物すべてをいくつかの箱に，どの箱の容量も越えないように詰め込みたい．使う箱の

‖ それぞれの問題の出典はいちいち示さない．NP 完全問題のバイブル: M.R.Garey and D.S.Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman and Company, 1979 を参照せよ．

個数をできるだけ少なくするには，どのように (どの品物をどの箱に) 詰め込んだらよいか？

(3) **整数線形計画法** (integer linear programming)　$m \times n$ 整数行列 A と $m \times 1$ 整数ベクトル $\mathbf{b}$ が与えられたとき，$A\mathbf{x} \leq \mathbf{b}$ を満たす $n \times 1$ 整数ベクトル $\mathbf{x}$ が存在するか？ なお，解 $\mathbf{x}$ を整数に限定せず，有理数の解を求める問題は $\mathbf{P}$ に属することが知られている**.

(4) **集合分割問題** (set partition)　整数の有限集合 $A \subseteq \mathbf{Z}$ が与えられたとき，$\sum_{x \in B} x = \sum_{x \in A-B} x$ を満たす $B \subseteq A$ が存在するか？

(5) **部分集合和問題** (subset sum)　$\mathbf{N}$ の有限部分集合 $A \subseteq \mathbf{N}$ と自然数 k が与えられたとき，$\sum_{x \in B} x = k$ を満たす $B \subseteq A$ が存在するか？

(6) **直和集合被覆問題** (exact cover)　有限集合 A と，A の部分集合の列 $S_1, \ldots, S_n$ が与えられたとき，$A = \bigcup_{i \in I} S_i$, $S_i \cap S_j = \emptyset$ $(i \neq j)$ を満たす $I \subseteq \{1, \ldots, n\}$ が存在するか? また，$S_1, \ldots, S_n$ の中から最小個数の $S_{i_1}, \ldots, S_{i_k}$ を選び，A のどの元も $S_{i_1}, \ldots, S_{i_k}$ のどれかに属すようにせよ，という問題 (**最小集合被覆問題**) も $\mathbf{NP}$ 完全である．両問題とも，すべての S_i の元の個数が 3 以下であっても $\mathbf{NP}$ 完全であるが，2 以下なら $\mathbf{P}$ に属す.

(7) **ジグソーパズル問題** (zigsaw puzzle)　多角形の組 $P_1, \ldots, P_n$ が与えられたとき，これらを平面上に隙間なく，かつ重なることなく並べて正方形とすることができるか？ 多角形は頂点の座標の組で与える.

(8) **時間割作成問題** (time table)　先生の集合 T，科目の集合 S，授業設置可能時間帯の集合 H，各先生 $t \in T$ が使える時間帯の集合 $A(t) \subseteq H$，各科目 $s \in S$ の開講可能な時間帯の集合 $A(s) \subseteq H$，各先生 t が受け持たなければならない科目 s のクラス数 $R(t, s)$ が与えられたとき (集合はすべて有限集合)，これらの条件を満たす授業時間割を作成することができるか？

(9) **スケジューリング問題** (scheduling)　仕事の集合 T，各仕事 $t \in T$ を仕上げるのにかかる時間 $C(t)$，各仕事のデッドライン (締切日) までに残された時間 $D(t)$，どの仕事にも同じ条件で使える機械の台数 m が与えられたとき，これら m 台の機械に各仕事を実行させる順序をうまく割り当てて，すべての仕事をデッドラインまでに仕上げることができるか？

(10) **最長道問題** (longest path)　グラフ $G = (V, E)$ と 2 頂点 $s, t \in V$ と整数 k が与えられたとき，s と t を結ぶ長さが k 以上の道が存在するか否かを判定せよ．閉路のない有向グラフの場合には $\mathbf{P}$ に属す．G に長さ k 以下の道が存在するか否かを判定する問題も $\mathbf{P}$ に属す.

(11) **部分グラフ同型問題** (subgraph isomorphism)　2 つのグラフ G_1, G_2 が

**L.G.Khachiyan, A polynomial time algorithm for linear programming, *Dokl. Akad. Nauk SSSR* 244, pp.1093-1096, 1979.

与えられたとき，G_1 が G_2 の部分グラフであるか否か判定せよ．

(12) **3 彩色問題** (3-colorability)　グラフ G の k 彩色とは，隣接する頂点の色が異なるように，G のすべての頂点を k 色以内で色付けする方法のことである．G が 3 彩色可能かどうか判定せよ．3 彩色問題は平面グラフ (平面上に辺が交差することなく描かれたグラフ) に限定しても **NP** 完全である．どの頂点の次数も 3 以下である場合には **P** に属す．2 彩色問題は **P** に属す．

(13) **頂点被覆問題** (vertex cover)　グラフ $G = (V, E)$ の頂点被覆とは，$uv \in E$ なら $u \in A$ または $v \in A$ であるような集合 $A \subseteq V$ のことである．$|A|$ をサイズという．グラフ G が与えられたとき，G の頂点被覆でサイズが最小のものを求めよ．

(14) **独立集合問題** (independent set)　$G = (V, E)$ をグラフとする．$U \subseteq V$ のどの 2 点 u, v に対しても $uv \notin E$ であるならば U は G の独立集合であるという．G の独立集合で頂点数が最大のものを求めよ．極大な独立集合を求める問題は **P** に属す (7 章の演習問題 9 参照)．

(15) **クリーク問題** (clique)　完全部分グラフのことを**クリーク**という．グラフ G が与えられたとき，その最大クリークを求めよ．G が三角形 (頂点数 3 のクリーク) を含むか否かを判定する問題は **P** に属す．

(16) **全域部分グラフ問題** (spanning subgraph)　グラフ $G = (V, E)$ が与えられたとき，G の全域部分グラフ ($G' = (V, E')$, $E' \subseteq E$ であるもの) で平面グラフであるものが存在するか否かを判定せよ．また，G と整数 k が与えられたとき，k 重連結であるような最小の G の全域部分グラフを見つける問題，2 部グラフであるような最大の全域部分グラフを見つける問題，どの頂点の次数もたかだか k であるような最大の全域連結部分グラフを見つける問題，はいずれも **NP** 完全である．

(17) **最小共通語問題** (shortest common superstring)　アルファベット Σ 上の文字列 $w_1, \ldots, w_n$ が与えられたとき，これらすべてを部分語として含むような，長さが最小の文字列を求めよ．$|\Sigma| = 2$ でも **NP** 完全である．すべての w_i の長さが 2 以下であるなら **P** に属す．

(18) **2 次合同式** (quadratic congruence)　正整数 a, b, c が与えられたとき，$x^2 \equiv a \pmod b$ の正整数解 $x < c$ が存在するか？ 類似の問題「$ax^2 + by = c$ の正整数解 x, y が存在するか？」も **NP** 完全である．

これらの問題が **NP** 完全であることは，定理 5.5, 5.6 の証明と同様に，すでに **NP** 完全であることが分かっている問題をその問題に還元することによって証明する．例えば，3SAT $\leq_{poly}$ クリーク問題 $\leq_{poly}$ 頂点被覆問題 $\leq_{poly}$ 部分集合和問題 であるが，その証明の概略は次のようなものである．

(3SAT $\le_{poly}$ クリーク問題) 3-CNF $\phi = C_1 \wedge \cdots \wedge C_k$ に対し，グラフ G_ϕ を次のように構成する．各節 $C_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$ ごとに v_{i1}, v_{i2}, v_{i3} を G_ϕ の頂点とし，$i \neq j$ かつ ℓ_{ir} と ℓ_{js} が互いに他の論理否定でないとき v_{ir} と v_{js} を辺で結ぶ．このとき，ϕ が充足可能である必要十分条件は G_ϕ がサイズ k のクリークを持つことである．

(クリーク問題 $\le_{poly}$ 頂点被覆問題) グラフ G と自然数 k に対し，グラフ $\overline{G}$ (G の**補グラフ** (complement of G) という) を次のように構成する: $(u,v) \in E(\overline{G}) \iff (u,v) \notin E(G)$．このとき，$G$ がサイズ k のクリークを持つ必要十分条件は $\overline{G}$ がサイズ $|V(G)| - k$ の頂点被覆を持つことである．

(頂点被覆問題 $\le_{poly}$ 部分集合和問題) グラフ $G = (V, E)$ と自然数 k に対して，集合 S を次のように構成する．頂点 v が辺 e の一方の端点であるとき，v と e は**接続** (incident) しているという．$|V| \times |E|$ 行列 $B = (b_{ij})$ を

$$b_{ij} = \begin{cases} 1 & \text{頂点 } v_i \text{ と辺 } e_j \text{ が接続しているとき} \\ 0 & \text{そうでないとき} \end{cases}$$

と定義する．頂点 v_i に対応して自然数 s_i を，$s_i = 4^{|E|} + \sum_{j=1}^{|E|} b_{ij} 4^{j-1}$ と定義する (B の第 i 行を 4 進数として見たもの)．また，辺 e_j に対応して自然数 t_j を $t_j = 4^j$ と定義する．

$$S = \{s_i, t_j \mid 1 \le i \le |V|,\ 1 \le j \le |E|\}, \quad s' = k4^{|E|} + \sum_{j=1}^{|E|} 2 \cdot 4^{j-1}$$

とおく．このとき，G がサイズ k の頂点被覆を持つ必要十分条件は $\sum_{x \in S'} x = s'$ を満たす $S' \subseteq S$ が存在することである．

問 5.6 (1) 上記の各還元をきちんと証明せよ．
(2) 集合分割問題 $\le_{poly}$ 部分集合和問題 であることを証明せよ．

NP 完全問題は多項式時間還元によって互いに他へ変換できるという意味で非常に密接な関係にある．しかし，実はもっと強いことが成り立つことが知られている．すなわち，現在知られている **NP** 完全問題はすべて互いに多項式同型である．ただし，2 つの言語 $L, L' \subseteq \Sigma^*$ が**多項式同型** (polynomially isomorphic) であるとは，次の条件を満たす関数 $h : \Sigma^* \to \Sigma^*$ が存在することである:

(i) h は単射 (1 対 1) である．
(ii) 任意の $x \in \Sigma^*$ に対して，$x \in L \iff h(x) \in L'$.
(iii) h もその逆関数 h^{-1} も多項式時間で計算可能である．

言語 $L \subseteq \Sigma^*$ に対して

$$\mathrm{dense}_L(n) = |\{x \in L \mid |x| \le n\}|$$

と定義する．$\mathrm{dense}_L(n)$ はたかだか指数関数であるが，$\mathrm{dense}_L(n) = n^{O(1)}$ (たかだか n の多項式) であるとき L は**疎** (sparse) であるといい，そうでないとき**密** (dense) であるという．

問 5.7 (1) tally 言語 (1 文字アルファベット上の言語) は疎な言語であることを示せ．また，密な正規言語が存在することを示せ．
(2) L_{SAT} は密な言語であることを示せ．

補題 5.4 L と L' が多項式同型であるなら，$\mathrm{dense}_L(n) = \mathrm{dense}_{L'}(p(n))$ かつ $\mathrm{dense}_{L'}(n) = \mathrm{dense}_L(q(n))$ となる多項式 p, q が存在する．

証明 容易 (演習問題とする)． □

現在知られている **NP** 完全問題の言語はすべて密である (一般に証明されているわけではない)．したがって，補題 5.4 より，それらは疎な言語と多項式同型ではありえない．実際，次のことが成り立つ[††]:

命題 5.1 **NP** 完全な tally 言語が存在するならば $\mathbf{P} = \mathbf{NP}$ である．

5.2.2 クラス co-NP

NP が補集合で閉じているかどうかは現在のところ未解決の問題である．**P**, **NP**, co-**NP**, **PSPACE** の間には図 5.9 に示したような包含関係があるが，これらの包含関係が真であるか否かはひとつとして分かっていない．**NP** と co-**NP** とは鏡に映したように非常に密接な関係があるから，**NP** について知るためにも，co-**NP** について知ることには意味がある．

定理 5.7 言語 L が **NP** 完全なら，その補集合 L^c は co-**NP** 完全である．

証明 容易 (読者への演習問題とする)． □

例 5.2 (1) ブール式 ϕ がトートロジーである必要十分条件は，$\neg\phi$ が充足可能でないことであるから，与えられたブール式がトートロジーであるかどうかを判定する問題は co-**NP** 完全である．

(2) 与えられたグラフがハミルトン閉路を持たないかどうかを判定する問題は co-**NP** 完全である．この問題は **NP** に属すかどうか知られていない．ハミ

[††] 多項式同型という概念は L.Berman and J.Hartmanis, Isomorphism and density of NP and other complete sets, *SIAM J. Comput.* 6, pp.305-322, 1977 で導入された．命題 5.1 は P.Berman, Relationship between the density and deterministic complexity of NP-complete languages, *Proc. 5th Intern. Colloq. on Automata, Languages, and Programming* (*Lecture Notes in Comput. Sci.* 62), pp.63-71, 1978 による．

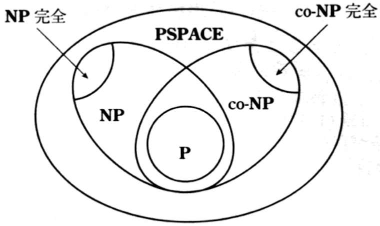


図 5.9　**P**, **NP**, co-**NP** の関係

ルトン閉路を "持つかどうか" は非決定性の多項式時間アルゴリズムで容易に判定できるのに, "持たないかどうか" を判定することはそれほど容易ではないようである (7.2 節参照).　□

co-**NP** に属す問題の中でもっとも重要なものは**素数判定問題** (problem of primality) PRIME である. 整数 $n \in \{0,1\}^*$ (2 進数表現) が素数かどうかを判定するための自明なアルゴリズム ($2 \sim \sqrt{n}$ の範囲の整数で n を割ってみる) は $O(\sqrt{n}\log^2 n)$ 時間アルゴリズムであるが, これは n の長さ ($\log_2 n$) の多項式ではない. しかし実は, 素数かどうかの判定も, 素数でない (合成数) かどうかの判定も, **NP** に属すことが証明できる. 以下で, PRIME $\in$ co-**NP** を証明する.

補題 5.5 (フェルマーの小定理) $n \geq 2$ が素数ならば, $0 < a < n$ なる任意の整数 a に対して $a^{n-1} \equiv 1 \pmod{n}$ である.

証明　整数 x が整数 y を割り切ることを $x \mid y$ で表わす. $0 < i < n$ に対して $m_i = ia$ とおく. 任意の相異なる i, j に対して $m_i \not\equiv m_j \pmod{n}$ でなければならない. なぜなら, もしそうでないとすると $n \mid (i-j)a$ となり, i, j, a のどれも n より小さいという仮定に反す. 同様に, $m_i \not\equiv 0 \pmod{m}$ でなければならない. したがって, $m_1, m_2, \ldots, m_{n-1}$ を適当に並びかえると, 法 m のもとで $1, 2, \ldots, n-1$ に等しい. よって,

$$a^{n-1}\prod_{i=1}^{n-1} i = \prod_{i=1}^{n-1} m_i \equiv \prod_{i=1}^{n-1} i \pmod{n},$$

すなわち

$$(a^{n-1}-1)\prod_{i=1}^{n-1} i \equiv 0 \pmod{n}$$

が成り立つ. $n \not| \prod_{i=1}^{n-1} i$ であるから $n \mid (a^{n-1}-1)$ でなければならない.　□

整数 a と b の最大公約数を $\gcd(a,b)$ で表わすと

$$\gcd(a,b) = \begin{cases} a & b=0\text{のとき} \\ b & a=0\text{のとき} \\ \gcd(a-b,b) & a \geq b > 0\text{のとき} \\ \gcd(a,b-a) & b \geq a > 0\text{のとき} \end{cases}$$

であることが容易にわかる．$a \geq b > 0$ の場合，$\gcd(a,b) = \gcd(a-b,b)$ を $a > b$ である間つづけて行なうことによって $\gcd(a,b) = \gcd(a \ \mathbf{mod}\ b, b)$ が得られる（$a \ \mathbf{mod}\ b$ は a を b で割った商を表わす）．$b \geq a > 0$ の場合も同様である．よって，$\gcd(b,a) = \gcd(a,b)$ であることと，$a < b$ ならば $a \ \mathbf{mod}\ b = a$ であることを考慮すると，$\gcd(a,b) = \gcd(b, a \ \mathbf{mod}\ b)$ であることが導かれる．この事実に基づくのが有名な**ユークリッドの互除法**である：

```
function Euclid(a, b)
begin
    if b = 0 then return a else return Euclid(b, a mod b) end
end
```

$f_0 = 0$，$f_1 = 1$，$n \geq 1$ のとき $f_{n+1} = f_n + f_{n-1}$，により定義される数列 $\{f_n\}_{n\geq 0}$ を**フィボナッチ数列**(Fibonacci sequence) という．

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

であることが帰納法で証明できる．次の補題も帰納法で示すことができる．

補題 5.6 $a > b \geq 0$ とする．Euclid(a,b) の再帰呼び出しの最大深さが $k \geq 1$ ならば $a \geq f_{k+2}$ かつ $b \geq f_{k+1}$ である．

補題5.6 は，$\gcd(a,b)$ が（a,b の2進数表現の長さの）多項式時間で求められることを示している．

次に，a,b が与えられたとき，$ax + by = \gcd(a,b)$ を満たす x,y を求めるように上記のアルゴリズムを拡張する（$a \ \mathbf{div}\ b$ は a を b で割った商を表わす）：

```
function extended_Euclid(a, b)
begin
    if b = 0 then return (a, 1, 0)
    else
        (c, x1, y1) ← extended_Euclid(b, a mod b);
        x ← y1; y ← x1 − (a div b)y1;
        return (c, x, y)
    end
end
```

問 5.8 extended_Euclid(a,b) によって求めた (c,x,y) は $ax + by = c = \gcd(a,b)$ を満たしていることを証明せよ．

したがって，$\gcd(a,b) \mid c$, $c = \gcd(a,b)d$ であるとき，$ax_0 + by_0 = \gcd(a,b)$ を満たす (x_0, y_0) を求め $x = dx_0$, $y = dy_0$ とおくと，(x, y) は $ax + by = c$ の整数解である．逆に，$ax + by = c$ が整数解を持つためには $\gcd(a,b) \mid c$ が成り立つことが必要であることは明らかであるから，次の補題を得る:

補題 5.7 $ax+by = c$ が整数解 (x, y) をもつための必要十分条件は $c \mid \gcd(a,b)$ が成り立つことである

系 5.3 $\gcd(a,n) = 1$ である必要十分条件は，$ab \equiv 1 \pmod n$ を満たす b が存在することである．

証明 補題 5.7 より，

$$\begin{aligned}\gcd(a,n) = 1 \quad &\Longleftrightarrow \quad ab + nc = 1 \text{ を満たす } (b,c) \text{ が存在する}\\ &\Longleftrightarrow \quad n \mid (1 - ab) \text{ を満たす } b \text{ が存在する}\\ &\Longleftrightarrow \quad ab \equiv 1 \pmod n \text{ を満たす } b \text{ が存在する}\end{aligned}$$

である． □

補題 5.8 $n \geq 2$ が合成数なら，$a^{n-1} \not\equiv 1 \pmod n$ を満たす整数 $0 < a < n$ が存在する．

証明 n が合成数なら $\gcd(a,n) \neq 1$ を満たす整数 $1 < a < n$ が存在する (a としては n の約数をとればよい)．$a^{n-1} \equiv 1 \pmod n$ と仮定する．$b = a^{n-2}$ とすると $ab \equiv 1 \pmod n$ であるから，系 5.3 より $\gcd(a,n) = 1$ となり，矛盾．
□

補題 5.5, 5.8より，次の定理が得られる:

定理 5.8 $n \geq 2$ が素数である必要十分条件は，$0 < a < n$ なる任意の a に対して $a^{n-1} \equiv 1 \pmod n$ が成り立つことである．

この事実は素数性の判定に使うことができる．$0 < a < n$ なる整数 a に対して，$a^{n-1} \equiv 1 \pmod n$ が成り立つか否かを判定することを**フェルマーテスト** (Fermat test) と呼ぶ．しかし，このような判定をしなければならない a は $n - 1 = O(2^{n\text{の桁数}})$ 個もあるので，(n の桁数の) 多項式時間ではできない．

定理 5.9 PRIME $\in$ **NP** $\cap$ co-**NP**.

証明 PRIME $\in$ co-**NP** だけ証明する．PRIME $\in$ **NP** の証明は脚注に示した Pratt の論文を参照されたい§．次の NTM M を考える:

§V.R.Pratt, Every prime has a succinct certificate, *SIAM J. Comput.* 4, pp.214-220, 1975.

procedure $M(n)$
begin
　$a \in \{1, 2, \ldots, n-1\}$ を推測する;
　if $a^{n-1} \not\equiv 1 \pmod{n}$ **then** 入力を受理する **else** 拒否する **end**
end

$a^{n-1} \bmod n$ は $O(\log^2 n)$ 時間で計算できる (問 5.9) ので，M は多項式時間限定である．定理 5.8 より，M が受理する n は合成数である． □

問 5.9 a^n を計算する $O(\log^2 n)$ 時間アルゴリズムを示せ．ヒント: a を平方することを $\log_2 n$ 回行なえ．

素数性の判定に関しては，7 章で述べるように効率良い確率性アルゴリズムが存在するが，PRIME $\in$ **P** かどうかは未解決である．しかし，リーマン予想 (リーマンのゼータ関数と素数の分布に関する，整数論におけるもっとも有名な未解決問題) が正しいと仮定すると PRIME $\in$ **P** が成り立つことが証明されている．

P, **NP**, co-**NP** は，それぞれ異なるクラスであると信じられている．その意味で，co-**NP** 完全な問題は **P** にもっとも属しそうもない問題であるが，どうやら **NP** にもっとも属しそうもない問題でもあるようである (**NP** = co-**NP** だとすると **P** = **NP** となるが，**P** $\neq$ **NP** であろうと強く予想されている):

定理 5.10 L が co-**NP** 完全で $L \in$ **NP** なら **NP** = co-**NP** である．

証明 L は co-**NP** 完全だから，$L' \in$ co-**NP** とすると，$L' \leq_{poly} L$ via N となる多項式時間限定変換機 N が存在する．$L \in$ **NP** であることより，L を受理する多項式時間限定 NTM M が存在する．NTM M' を次のように構成する．入力 x に対して，まず N を模倣して $N(x)$ を出力する．これを M に入力し，M が $N(x)$ を受理 (拒否) したら M' は x を受理 (拒否) する．明らかに M' は多項式時間限定であり，

$$x \in L(M') \Longleftrightarrow N(x) \in L(M) = L \Longleftrightarrow x \in L'$$

であるから $L(M') = L'$ である．よって，co-**NP** $\subseteq$ **NP**. 逆の包含関係は **NP** と co-**NP** の対称性から導かれる． □

5.2.3 NP 完全問題と暗号系

かつては，暗号というと戦時中の秘密通信や犯罪活動など，暗いイメージを伴ったものであったが，現在のように情報システムが発達し，大量のデータが

G.L.Miller, Riemann's hypothesis and test for primality, *J. Comput. Sys. Sci.* 13, pp.300-317, 1976.

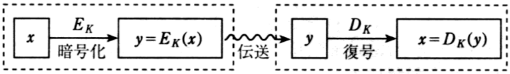


図 5.10 暗号系

蓄積されネットワーク上を伝送される時代になり，暗号の必要性が再認識され，ポジティブな活用に向けて積極的な研究が行なわれるようになった．その現代暗号学は計算量理論抜きには語れない．

古典的な暗号系 (cryptosystem) のイメージによると (図 5.10 参照)，A から B へ送りたいメッセージ (まだ暗号化してないメッセージを**平文** (plain text) という) x を誰かが途中で奪取あるいは盗聴しても解読できないように，A は x を何らかの方法で**暗号化** (encript) した**暗号文** (cryptotext) $E_K(x)$ を B に送る．A は暗号化にあたって**鍵** (key) と呼ばれるもの K を使い，この鍵は別のルートで B に渡される (あるいは，前もって A から直接 B に渡しておく)．B は暗号文 $E_K(x)$ を受け取ると，鍵 K を使ってこれを**復号化** (decrypt) して元のメッセージ $x = D_K(E_K(x))$ を得る．D_K は E_K の逆関数である．

良い暗号系とは，(1) 暗号化や復号化が簡単にできるように，E_K や D_K が容易に計算できる (すなわち，計算量の小さい) 関数であり，(2) 盗聴されても暗号文が解読されないように，復号化の方法 D_K を知らずに暗号文 $E_K(x)$ から x を計算することが困難なものでなければならない．

昔からある暗号系では，暗号化の方法 E_K とそのときに使う鍵 K は当事者 A, B 以外は知らないことが前提になっているのが普通であり，もしこれらが分かると復号化の方法 D_K もすぐにわかってしまった．これに対し，**公開鍵暗号系** (public-key cryptography) * では E_K と K を公開する．それでも通信が漏れたことにならないためには，E_K が公開されていても，その逆関数 D_K の計算が手に負えないほど難しいものであればよい．この E_K のような性質を持つ関数を**一方通行関数** (one-way function) という．正確には，次の条件を満足する関数 $f : \Sigma^* \to \Delta^*$ を一方通行関数という (Σ と Δ は有限アルファベット)：

(i) f は単射 (1 対 1) である．

(ii) 任意の $x \in \Sigma^*$ に対し， $|x|^{1/k} \le |f(x)| \le |x|^k$ を満たす正整数 k が存在する．すなわち，$f(x)$ の長さと x の長さは大きく違わない (互いに他の多項式程度の違いしかない)．

(iii) f は多項式時間で計算できる．

---

*W.Diffie and M.E.Hellman, New directions in cryptography, *IEEE Trans. Inform. Theory* IT-22, pp.644-654, 1976.

(iv) f の逆関数 f^{-1} を計算する多項式時間アルゴリズムは存在しない．すなわち，y を入力したとき，$f(x)=y$ である x を出力するか，そのような x が存在しない場合には "no" を出力する，多項式時間アルゴリズムは存在しない．

このような一方通行関数が存在するかどうかは $\mathbf{P}=\mathbf{NP}$ 問題と深い関係がある．任意の入力に対したかだか 1 つの受理計算しか存在しないような NTM を**無曖昧 NTM** (unambiguous NTM) という．多項式時間限定の無曖昧 NTM が受理する言語のクラスを **UP** で表わす (unambiguous P または unique P と読む)．明らかに，$\mathbf{P}\subseteq\mathbf{UP}\subseteq\mathbf{NP}$ である．

命題 5.2 一方通行関数が存在する $\Longleftrightarrow$ $\mathbf{P}\neq\mathbf{UP}$．[†]

$\mathbf{P}\subsetneq\mathbf{UP}\subsetneq\mathbf{NP}$ であろうと予想されているが証明されたわけではないので，一方通行関数が存在するかどうかはまだ分かっていない．しかし，一方通行関数の候補はいくつか知られている．例えば，非常に大きな 2 つの素数の積をとる関数 $f(p,q)=pq$ とか，素数を法としてべき乗をとる関数 $g(r,x,p)=r^x \bmod p$ などである．これらをもとに提案されたのが **RSA システム** (RSA public-key cryptosystem) である[‡]．これは，2 つの大きな素数 (例えば，10 進数で 100 桁以上) の積 $n=pq$ を素因数分解して p,q を復元することはほとんど不可能であろうという予想に基づいている (5.2.2 節および演習問題 17 参照)．その一方，大きな素数をランダムに生成する効率良いアルゴリズムが存在する．

$\phi(pq)=pq(1-1/p)(1-1/q)$ とし (ϕ はオイラー関数 (Euler function) と呼ばれる関数で，整数論では重要な関数である: $\phi(n)=$「n と互いに素な, n 以下の自然数の個数」)，$e>1$ を $\phi(pq)$ と互いに素な整数とする．暗号の受け手 B は p,q を知っており，積 pq と e を公開しておく (**公開鍵**)．送り手 A は公開鍵を使って，メッセージ x を

$$y=x^e \ \mathbf{mod}\ pq$$

と暗号化する．B は，合同式

$$ed\equiv 1 \pmod{\phi(pq)}$$

を満たす整数 d を計算して求める．e が $\phi(pq)$ と互いに素であることより，d を求める Euler のアルゴリズムと呼ばれる高速のアルゴリズムが存在する (整数論の知識を必要とするのでここでは詳細を省略するが，$O(\log^3 pq)$ 時間で計算できる)．B は

$$y^d=x^{ed}=x^{1+k\phi(pq)}=x \ \mathbf{mod}\ pq$$

を計算することにより x を復元することができる．

[†] J.Grollman and A.L.Selman, Complexity measures for public-key cryptography, *SIAM J. Comput.* 17, pp.309-335, 1988.

[‡] R.L.Rivest, A.Shamir and L.Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Commun. Assoc. Comput. Mach.* 21, pp.120-126, 1978.

5.3 P と NL：実際的計算可能問題のクラス

前節で述べたように，**P** は実際的な意味で計算可能な問題のクラスと考えることができる．この節では **P** およびその部分クラスである **NL** について考察する．

5.3.1 クラス P

P 完全問題の例として論理回路値問題を考える．はじめに，論理回路とは何かを定義する．例えば，ブール式

$$((0 \vee x_1) \vee \neg x_2) \wedge (\neg x_2 \vee (x_2 \wedge x_3))$$

を考えよう．このブール式は，図 5.11 のように閉路のない有向グラフで表わすことができる (図では矢印を示していないが，辺は上から下に向かっているものとする)．これを論理回路という．各頂点には入出力あるいは論理演算を行なうための素子が置かれている．入次数が 0 の頂点 ⓪，$\textcircled{x_1}$，$\textcircled{x_2}$，$\textcircled{x_3}$ は論理定数または論理変数 (ブール変数) を表わし，入力ノードと呼ばれる．出次数が 0 の頂点が 1 つだけあり (図 5.11 では v_0)，これは出力ノードと呼ばれる．入出力ノード以外の各頂点には論理演算 $\vee, \wedge, \neg$ のどれか一つがラベル付けされている．$\vee, \wedge, \neg$ がラベル付けされている頂点をそれぞれ OR ゲート，AND ゲート，NOT ゲートと呼ぶ．OR ゲートと AND ゲートの入次数は 2，NOT ゲートの入次数は 1 でなければならない (このように，ブール式を表わす論理回路では出力ノードは 1 つだけであり，論理演算も $\neg, \wedge, \vee$ だけに限定されているが，一般の論理回路では出力ノードが複数あってもよいし，$\neg, \wedge, \vee$ 以外の論理演算があってもよい．また，入次数が 2 以下，出次数が 1 という制約もない)．

以上のことをもとに，論理回路とは何かを次のように形式的に定義する．n **入力** m **出力の論理回路** (Boolean circuit) とは，閉路を持たないラベル付き有向グラフ

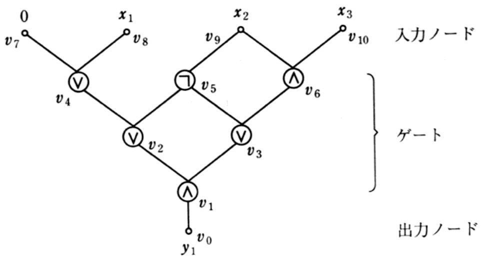


図 **5.11** 論理回路

$$C = (V, E),\quad \tau : V \to \Omega \cup \{0, 1, x_1, \ldots, x_n, y_1, \ldots, y_m\}$$

のことである．ただし，Ω は回路に使われる基本素子の集合であり，C の**基底** (basis) という．任意のブール関数 $f : \{0,1\}^\ell \to \{0,1\}$ $(\ell \geq 1)$ を基本素子としてよいが，通常は基本素子としては $\{\neg, \wedge, \vee\}$ などを考える．$v \in V$ に対し $\tau(v)$ を v の**タイプ** (type) という．$v \in V$ の入次数を**ファンイン** (fan-in)，出次数を**ファンアウト** (fan-out) という．C は次の条件を満足していなければならない．

1. $\tau(v) = 0$ または $\tau(v) = 1$ なる頂点 v を**定数ノード**といい，$\text{in-deg}(v) = 0$ でなければならない．
2. 各 i $(1 \leq i \leq n)$ に対し $\tau(v) = x_i$ なる頂点 v がちょうど 1 つあり，$\text{in-deg}(v) = 0$ である．v を**入力ノード** (input node) といい，x_i を**入力変数** (input variable) と呼ぶ．以後，v と x_i を同一視する．
3. 各 i $(1 \leq i \leq m)$ に対し $\tau(v) = y_i$ なる頂点 v がちょうど 1 つあり，$\text{in-deg}(v) = 1$, $\text{out-deg}(v) = 0$ である．v を**出力ノード** (output node) といい，y_i を**出力変数** (output variable) と呼ぶ．以後，v と y_i を同一視する．
4. $\tau(v) \in \Omega$ なる頂点 v を**ゲート** (gate) といい，$\tau(v)$ が ℓ 変数ブール関数なら $\text{in-deg}(v) = \ell$ でなければならない．例えば，$\tau(v) = \neg$ ならば $\text{in-deg}(v) = 1$ である．ファンアウトに制限はない．$\tau(v) = \neg$ なる v を **NOT ゲート**，$\tau(v) = \wedge$ なる v を **AND ゲート**，$\tau(v) = \vee$ なる v を **OR ゲート**という．

C の入力変数 $x_1, \ldots, x_n$ を明示するときは $C(x_1, \ldots, x_n)$ と書く．C の頂点の個数を**サイズ** (size) といい $\text{size}(C)$ で表わす．入力ノードからの距離の最大値をそのゲート (あるいはノード) の**深さ** (depth) といい，出力ノードの深さ (出力ノードが複数ある場合にはその最大値) を論理回路 C の深さという．C の深さを $\text{depth}(C)$ で表わす．

例えば，図 5.11 の論理回路 $C(x_1, x_2, x_3)$ において，

$$\tau(v_8) = x_1,\ \tau(v_7) = 0,\ \ldots,\ \tau(v_4) = \vee,\ \ldots,\ \tau(v_0) = y_1$$

であり，v_7 は定数ノード，$v_8 \sim v_{10}$ は入力ノード，v_0 は唯一の出力ノード，$v_1 \sim v_6$ はゲートである．この論理回路のサイズは 11，深さは 4 である (6.2 節の問 6.3 参照)．

さて，論理回路 $C(x_1, \ldots, x_n)$ があり，C の入力変数 $x_1, \ldots, x_n$ それぞれに $\sigma(x_i) \in \{0,1\}$ が入力として与えられたとき $(1 \leq i \leq n)$，C における**計算**は次のように行なわれる．各頂点 v における値は $\sigma : \{x_1, \ldots, x_n\} \to \{0,1\}$ に依存して定まるものなので，$\hat{\sigma}(v)$ と表わすことにする．

1. $\tau(v)=0$ ならば $\hat{\sigma}(v)=0$.
2. $\tau(v)=1$ ならば $\hat{\sigma}(v)=1$.
3. $\tau(v)=x_i$ ならば $\hat{\sigma}(v)=\sigma(x_i)$.
4. $\tau(v)\in\Omega$ が ℓ 変数ブール関数で，$\{u \mid (u,v)\in E\}=\{u_1,\ldots,u_\ell\}$ とする．$\hat{\sigma}(u_1),\ldots,\hat{\sigma}(u_\ell)$ がすでに定まっていたら $\hat{\sigma}(v)=\tau(v)(\hat{\sigma}(u_1),\ldots,\hat{\sigma}(u_\ell))$.
5. $\tau(v)=y_i$, $(u,v)\in E$ で $\hat{\sigma}(u)$ がすでに定まっていたら $\hat{\sigma}(v)=\hat{\sigma}(u)$.

上記の 4 は，ゲート v における計算結果 $\hat{\sigma}(v)$ は，$(v,w)\in E$ なるすべての w に同じ値で伝えられることを示している．

$\hat{\sigma}(y_i)$ は**入力** $(\sigma(x_1),\ldots,\sigma(x_n))$ に対する，出力ノード y_i における**出力**であり，$(\hat{\sigma}(y_1),\ldots,\hat{\sigma}(y_m))$ を $C(\sigma(x_1),\ \ldots,\ \sigma(x_n))$ とも書く．このように，n 入力 m 出力の論理回路 $C=C(x_1,\ldots,x_n)$ は $\{0,1\}^n$ から $\{0,1\}^m$ への関数 (n **入力** m **出力のブール関数**という) を定める．$m=1$ (出力変数が 1 つだけ)，$\Omega=\{\neg,\wedge,\vee\}$ に制限した論理回路 C は 1 つのブール式を表わし，ブール関数 $\{0,1\}^n\to\{0,1\}$ を定める．とくに，入力変数を持たない場合 ($n=0$)，C の出力はつねに定数 0 か 1 である．

論理回路値問題 (Circuit Value Problem: **CVP** と略記する) とは，次のような決定問題のことである:

入力: 0 入力 1 出力の論理回路 C
出力: C の出力

ここで，CVP を言語として考える場合，C は $|\overline{C}|=O(\mathrm{size}(C)\log\mathrm{size}(C))$ を満たすようにアルファベット $\{0,1,\vee,\wedge,\neg,\#\}$ 上の語 $\overline{C}$ として符号化する (簡潔な符号化: 2.3 節の例 2.5 参照)．

定理 5.11 CVP は $\mathbf{P}$ 完全である．

証明 論理回路 C と C への入力が与えられたら，その出力を C のサイズの多項式時間で計算できることは明らかであろう．よって，CVP は $\mathbf{P}$ に属す．

CVP が $\mathbf{P}$ 完全であることを示すために，任意の $L\in\mathbf{P}$ が CVP に対数領域還元可能であることを示そう．L を受理する $p(n)$ 時間限定 DTM を M とする ($p(n)$ は多項式)．M への任意の入力 x に対し，

$$x\in L(M)\Longleftrightarrow C_x=1$$

が成り立つような，0 入力 1 出力の論理回路 C_x を構成するための対数領域アルゴリズムを示せばよい．以下で作る論理回路はいずれも入力変数を持たないものであり，C が論理回路のとき $C=1$ と書いたら，C の出力が 1 であることを表わすものとする．

M は単テープ TM (両方向無限テープ) であるとし，$M = (Q, \Sigma, \Gamma, \delta, q_1, F)$, $Q = \{q_1, \ldots, q_s\}$, $\Gamma = \{a_1, \ldots, a_m\}$, $a_1 = \flat$ とする．M の受理状態は 1 つだけで，一旦受理状態に入ったらそれ以降は何もしないまま動きつづけると仮定してもよい (すなわち，$F = \{q_s\}$, 任意の $a \in \Sigma$ に対し $\delta(q_s, a) = (q_s, a, 0)$ とする).

まず，C_x を構成するのに使う基本論理回路を作る．$x = a_{k_0} a_{k_1} \cdots a_{k_{n-1}}$ ($a_{k_i} \in \Sigma$) とする．M の動作開始時に第 i ($0 \le i \le n-1$) こまには a_{k_i} が，それ以外のこまには $\flat = a_1$ が書いてあり，ヘッドは第 0 こま上にある．また，動作中 M のヘッドはつねに第 $-p(n)$ こま 〜 第 $p(n)$ こまの間しか動かないことに注意する．$-p(n) \le i \le p(n)$, $1 \le j \le m$, $1 \le k \le s$, $0 \le t \le p(n)$ に対し，論理回路 $H(i,t)$, $A(i,j,t)$, $Q(k,t)$ を次の条件を満たすように作りたい．

- $H(i,t) = 1 \iff$ 時刻 t (M が動作を開始してから t ステップ目) に M のヘッドは第 i こま上にある．
- $A(i,j,t) = 1 \iff$ 時刻 t における M の第 i こまの内容は a_j である．
- $Q(k,t) = 1 \iff$ 時刻 t における M の状態は q_k である．

実際，H, A, Q は以下のように再帰的に定義すればよい．まず，時刻 0 においては

$$H(i,0) = \begin{cases} 1 & i = 0 \text{ のとき} \\ 0 & \text{その他}, \end{cases}$$

$$A(i,j,0) = \begin{cases} 1 & 0 \le i \le n-1 \text{ かつ } a_{k_i} = a_j \text{ のとき} \\ 1 & i < 0 \text{ または } n \le i \text{ で } j = 1 \text{ のとき} \\ 0 & \text{その他}, \end{cases}$$

$$Q(k,0) = \begin{cases} 1 & k = 1 \text{ のとき} \\ 0 & \text{その他} \end{cases}$$

とすればよい．

すでに，時刻 t における $H(i,t)$, $A(i,j,t)$, $Q(k,t)$ が定義されているものとすると，時刻 $t+1$ における M のヘッド位置を表わす論理式 $H(i,t+1)$ は次のように定義することができる:

$$\begin{aligned} H(i,t+1) \;=\; & \left[H(i-1,t) \wedge \left(\bigvee\nolimits_{\delta(q_k,a_j)=(*,*,1)} (A(i-1,j,t) \wedge Q(k,t)) \right) \right] \\ \vee\; & \left[H(i,t) \wedge \left(\bigvee\nolimits_{\delta(q_k,a_j)=(*,*,0)} (A(i,j,t) \wedge Q(k,t)) \right) \right] \\ \vee\; & \left[H(i+1,t) \wedge \left(\bigvee\nolimits_{\delta(q_k,a_j)=(*,*,-1)} (A(i+1,j,t) \wedge Q(k,t)) \right) \right]. \end{aligned}$$

1行目の $[\cdots]$ の部分は，「時刻 t において M のヘッドが第 $i-1$ こま上にあり，そこにテープ記号 a_j が書いてあり，そのときの M の状態が q_k で，遷移 $\delta(q_k, a_j) = (*, *, 1)$ が定義されている (すなわち，ヘッドが1こま右に動く) こと」が，時刻 $t+1$ において M のヘッドが第 i こま上に来るための十分条件の1つであることを表わしている．$\delta(q_k, a_j) = (*, *, 1)$ は，$*$ の部分が任意であることを略記したものである．2行目，3行目の $[\cdots]$ の部分は，ヘッドが動かない場合，1こま左に動く場合をそれぞれ表わしている．この3つのどれかが成り立つことが，時刻 $t+1$ において M のヘッドが第 i こま上に来るための必要十分条件である．

時刻 $t+1$ に M の第 i こまの内容が a_j であるための必要十分条件は，時刻 t に第 i こま上にヘッドがなかったためにそこで書き換えが行なわれなかったか，時刻 t にヘッドが第 i こま上にあり，そのこまの内容が $a_{j'}$，そのときの状態が q_k で，遷移 $\delta(q_k, a_{j'}) = (*, a_j, *)$ により $a_{j'}$ が a_j に書き換えられたか，のいずれかであるから，$A(i,j,t+1)$ は次のように定義すればよい:

$$\begin{aligned} A(i,j,t+1) \quad = \quad & [\neg H(i,t) \wedge A(i,j,t)] \\ & \vee \left[H(i,t) \wedge \left(\bigvee\nolimits_{\delta(q_k, a_{j'}) = (*, a_j, *)} (A(i,j',t) \wedge Q(k,t)) \right) \right]. \end{aligned}$$

同様に，$Q(k,t+1)$ は次のように定義すればよい:

$$Q(k,t+1) \quad = \quad \bigvee_{\substack{-p(n) \le i < p(n) \\ \delta(q_{k'}, a_{j'}) = (q_k, a_j, *)}} (Q(k',t) \wedge H(i,t) \wedge A(i,j',t)).$$

どの $H(i,t+1)$，$A(i,j,t+1)$，$Q(k,t+1)$ もサイズが $O(p(n))$ であり，$H(\cdot,t)$，$A(\cdot,\cdot,t)$，$Q(\cdot,t)$ から $O(p(n))$ 時間で作ることができる (証明は t $(0 \le t \le p(n))$ に関する帰納法)．

C_x を作るには，$H(i,0)$，$A(i,j,0)$，$Q(k,0)$ から始め，上記の再帰的定義通りに $H(i,t)$，$A(i,j,t)$，$Q(k,t)$ を結線して $H(i,t+1)$，$A(i,j,t+1)$，$Q(k,t+1)$ を作ることをすべての i,j,k $(-p(n) \le i \le p(n)$，$1 \le j \le m$，$1 \le k \le s)$ について順次行なう．x が M に受理される必要十分条件は，時刻 $p(n)$ に M が受理状態にいることであるから，

$$Q(s, p(n))$$

が求める C_x である．C_x のサイズが $O(p(n))$ であり，$O(\log n)$ 領域で x から構成できることは明らかであろう． □

定理 5.11 の証明において，C_x の構成にあたって NOT ゲートが使われているのは $A(i,j,t+1)$ の中の $\neg H(i,t)$ だけである．ところが，$\neg H(i,t)$ は $\bigvee_{\substack{-p(n) \le \ell \le p(n) \\ \ell \ne i}} H(\ell,t)$ と等しいので，これに置き換えると，C_x が NOT ゲート

を含まないように構成できる．したがって，論理回路を NOT ゲートを含まないものに制限 (**monotone CVP** と呼ぶ) しても CVP は **P** 完全のままである．

問 5.10 NAND ゲート (例 6.2 参照) しか含まない論理回路に制限しても CVP は **P** 完全のままであることを示せ．

その他の P 完全問題

NP 完全問題が何千と発見されているのと同様に，**P** 完全問題もたくさん発見されている[§]．**NP** 完全問題が **P** との関係において重要視されているのに対して，**P** 完全問題は，サイズが入力変数の多項式かつ深さが対数多項式であるような論理回路 (そのような論理回路で計算できる問題を **NC** 問題という) との関係で重要である．**P** と 論理回路との詳細な関係については 6.2 節で並列計算の観点から考察する．ここでは，典型的な **P** 完全問題を列挙しておこう[¶]．

(1) **有理線形計画法** (linear inequality)　$m \times n$ 整数行列 A と $m \times 1$ 整数ベクトル $\mathbf{b}$ が与えられたとき，$A\mathbf{x} \leq \mathbf{b}$ を満たす $n \times 1$ 有理ベクトル $\mathbf{x}{>}\mathbf{0}$ が存在するか？ ただし，$\mathbf{x}{>}\mathbf{0}$ は，$\mathbf{x}$ の成分がすべて非負で，かつ少なくとも 1 つ正の成分が存在することを表わす．$A\mathbf{x} \leq \mathbf{b}$ を $A\mathbf{x} = \mathbf{b}$ に置き換えた問題も **P** 完全である．

(2) **ライフゲーム** (life game)　時刻 0 において，格子平面上のいくつか (有限個) の格子点に石 (都市と考えよ) を置く．時刻 t に，格子点 (x, y) に隣接する 8 個の格子点に石が何個あるかに応じて，時刻 $t+1$ に格子点 (x, y) に石を置くか否かを次のルールに従って定める:

- 時刻 t に (x, y) には石が置いてなく，ちょうど 3 個の隣接点に石が置いてあったら，時刻 $t+1$ に (x, y) に石を置く (新しい都市の勃興)．
- 時刻 t に (x, y) には石が置いてあり，石が置いてある隣接点の個数が 2 または 3 以外だったら，時刻 $t+1$ に (x, y) の石を取り除く (適正密度を逸脱したことによる都市の滅亡)．
- 上記以外の場合は，時刻 t と $t+1$ における石の存否は不変．

時刻 0 に石を置く格子点の集合 S，時刻 t，1 つの格子点 (x, y) が与えられたとき，時刻 t に (x, y) に石が置いてあるか否かを判定せよ．

(3) **整数の mod 計算** (iterated mod)　2 進数 $a, b_1, \ldots, b_n$ が与えられたとき，$(\cdots((a \bmod b_1) \bmod b_2) \cdots \bmod b_n) = 0$ どうかを決定せよ．

[§] 最初の自然な P 完全問題は S.A.Cook, An observation on time-storage trade off, *J. Comput. System Sci.* 9, pp.308-316, 1974 で報告された．

[¶] 出典は省略するが，R.Greenlaw, H.J.Hoover and W.L.Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*, Oxford Univ. Press, 1995，あるいは，宮野悟:「並列アルゴリズム」，近代科学社，1993 に P 完全問題のリストがあるので参照せよ．

(4) **凸閉包** (convex hull)　$\mathbf{Q}$ を有理数すべての集合とする．$\mathbf{Q}^n$ の有限部分集合 S と $x \in \mathbf{Q}^n$ が与えられたとき，x が S の凸閉包に入っているかどうかを判定せよ．ただし，S の凸閉包とは S を含む最小の凸集合のこと，すなわち，任意の2点を結ぶ線分がその集合に入っているようなもののことである．

(5) **2項演算の下での生成可能性** (generatability)　有限集合 X とその上の2項演算 $\circ$，$S \subseteq X$，$x \in X$ が与えられたとき，S を含み $\circ$ で閉じている X の最小部分集合に x が含まれているかどうかを判定せよ．

(6) **パスシステム** (path system)　パスシステムとは4つ組 $P = (X, R, S, T)$ のことである．ただし，X は有限集合で，$R \subseteq X^3$，$S \subseteq X$，$T \subseteq X$ である．x が T に到達可能とは，$x \in T$ であるか，または $(x, y, z) \in R$ かつ y, z が到達可能なことである，と再帰的に定義する．T に到達可能な頂点が S に存在するか？

(7) **辞書式順序による底優先探索** (lexicographically-first DFS)　頂点に順序が付けられた無向グラフ $G = (V, E)$ と2つの頂点 $u, v \in V$ が与えられたとき，G を頂点の順序に従いながら DFS したとき u は v より先にたどられるか？

(8) **超単調増加ナップザック問題** (super increasing knapsack)　すべての i に対して $w_i > \sum_{j=1}^{i-1} w_j$ を満たすような整数の列 $w_1, \ldots, w_n$ と整数 W が与えられたとき，$W = \sum_{i \in I} w_i$ を満たす $I \subseteq \{1, \ldots, n\}$ が存在するか？

(9) **FF 型単調減少箱詰め問題** (first-fit decreasing bin packing)　$1 \geq a_1 \geq \cdots \geq a_n \geq 0$ を満たす有理数 $a_1, \ldots, a_n$ が与えられたとき，$a_1, \ldots, a_n$ をこの順序で，容量1の箱 $B_1, B_2, \ldots$ に順次，容量1を越えない範囲で詰め込めるだけいっぱい詰めていく．a_n を詰め終わったとき，各 a_i が何番目の箱に入っているかを求めよ．

(10) **文脈自由言語に関する所属問題**　文脈自由文法 G と語 x が与えられたとき，$x \in L(G)$ か否かを判定せよ．$L(G) = \emptyset$ か否かを判定する問題も，$L(G)$ が有限集合か否かを判定する問題も **P** 完全である．

5.3.2　クラス NL

DL $\subseteq$ **NL** $\subseteq$ **P** であるが，これらの包含関係が真であるかどうかは現在のところ未解決である．**DL** = **NL** かどうかに関して言うと，Savitch の定理から導かれる **NL** $\subseteq$ **DSPACE**$(\log^2 n)$ が現在知られている最良の結果である．

クラス **NL** の完全問題の例を次の定理で示す．どの **NL** 完全問題でもよいから，**P** に入らないことが証明できれば **NL** $\subsetneq$ **P** が導かれる．

定理 5.12 次の各問題は **NL** 完全である:

(1) GAP　(2) 2SAT

証明 (1) GAP が **NL** に属することは問 4.2 (4.1 節) で述べた．任意の $L \in \mathbf{NL}$ が GAP に対数領域還元可能であることを示せばよい．L を受理する $\log n$ 領域限定 NTM を M とする．入力 x に対する M の様相グラフを $G_{M,x}$ とする．初期 ID を表わす頂点を s とする．$G_{M,x}$ に新しい頂点 t を追加して，受理 ID を表わす頂点すべてから t へ向かう辺を加えてできるグラフを $G'_{M,x}$ とする．x から $G'_{M,x}$ を構成する $\log n$ 領域限定のアルゴリズムが存在する．

$$x \in L = L(M) \Longleftrightarrow G'_{M,x}\text{において } s \text{ から } t \text{ への道が存在する}$$

である．

(2) はじめに 2SAT $\in \mathbf{P}$ を示す．ϕ を 2-CNF のブール式とする．ϕ に現れるブール変数 x_i とその論理否定 $\neg x_i$ を頂点とし，$\neg x_i \vee x_j$ (または $x_j \vee \neg x_i$) が ϕ の節であるとき (x_i, x_j) を辺とする，有向グラフ G_ϕ を考える (図 5.12 に例を示した)．辺 (x_i, x_j) は論理式 $x_i \to x_j$ に対応している．したがって，x_i から x_j へ辺があれば $\neg x_j$ から $\neg x_i$ へも辺がある (対偶に相当する)．次のことを証明しよう．

補題 5.9 ϕ が充足不能である必要十分条件は，ϕ が次の条件を満たすブール変数 x を含んでいることである: x から $\neg x$ への道 (および $\neg x$ から x への道) が G_ϕ に存在する．

証明 そのような x と道が存在したとし，ϕ が充足可能であるとする．$\phi = 1$ とするような x の値が 1 であったとする (0 の場合も同様)．x から $\neg x$ への道を $x \to x_1 \to \cdots \to x_k \to \neg x$ とすると，$x = 1$ ならば $x_1 = \cdots = x_k = \neg x = 1$ でなければならない (なぜなら，$x \to x_1 = \neg x \vee x_1$ は ϕ の 1 つ節であり，$\phi = 1$ であるから $x \to x_1 = 1$ でなければならないが，$x = 1$ であるから $x_1 = 1$ である．以下同様にして，$x_2 = \cdots = x_k = \neg x = 1$ が導かれる)．これは矛盾である．

逆に，そのような x も道も存在しないとする．次のアルゴリズムに従って，G_ϕ の各頂点へ 0/1 を割り当てる (これによって，各変数への値の割り当てが定まる):

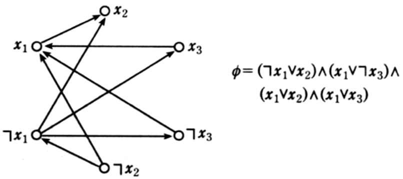


図 5.12　ϕ のグラフ G_ϕ

```
while まだ値が定まっていない Gφ の頂点 α が存在する do
    α ← 1;
    forall β do
        if α から β への道が存在する then β ← 1; ¬β ← 0 end
    end
end
```

α から β, $\neg\beta$ 両方へ道が存在することはない．なぜなら，もし存在したとすると，グラフ G_ϕ の対称性から，β, $\neg\beta$ から $\neg\alpha$ への道も存在することになり，x から $\neg x$ への道は存在しないという仮定に反す．また，α から β への道が存在する場合，β にすでに値 0 が割り当てられているということもない．なぜなら，もし β にすでに値が割り当てられていたとすると，α から β への道が存在することより，このステップ以前に α にはすでに値が割り当てられていることになってしまうからである．ゆえに，このアルゴリズムによって各変数の値は一意的に定まる．しかも，辺 $\alpha \to \beta$ (ϕ の 1 つの節を表わしている) に対する α の値が 1 ならば β の値も 1 であるから，$\alpha \to \beta$ の値は 1 である．よって，このような値の割り当てのもとで ϕ の値は 1 である．すなわち，ϕ は充足可能である．

以上で補題の証明が終わった．**NL** は補集合で閉じているので (定理 4.3)，2-CNF のブール式が充足不能であることを判定する $\log n$ 領域限定の NTM が存在することを示せばよいが，これは容易である． □

問 5.11 与えられた有向グラフが強連結 (すなわち，どの 2 頂点 u, v に対しても，u から v への道と v から u への道の両方が存在する) であるか否かを判定する問題は **NL** 完全であることが知られている．この問題が **NL** に属すことを示せ．

NL に属す問題が **NC** 問題であること (**NL** $\subseteq$ **NC** $\subseteq$ **P**) をはじめとして，**NL** と並列計算との関係については 6.2 節で詳細に考察する．対数領域還元性のもとではすべての **DL** の元が **DL** 完全になってしまうので，意味のある **DL** 完全問題を定義するためにはもっと厳しい還元性 (例えば，$\mathbf{NC}^1$ 還元) を用いる必要がある (6.2 節参照) [‖]．

5.4 その他の計算量のクラス

この節では **NP** を含む，より難しい問題のクラスについて考察する．

[‖] N.D.Jones, E.Lien and W.T.Laaser, New problems complete for deterministic log space, *Math. Systems Theory* 10, pp.1-17, 1976.
S.A.Cook and P.McKenzie, Problems complete for logarithmic space, *J. Algorithms* 8, pp.385-394, 1987.

5.4.1 PSPACE

1.5 節で量限定ブール式 (QBF) を定義した．例えば

$$\forall x(x \to x) = 1, \quad \forall x \forall y((x \to y) \to (y \to x)) = 0$$

のように，自由変数を含まない QBF の値はつねに 1 か 0 である．自由変数を含まない QBF が与えられたとき，その値が 0 であるか 1 であるかを決定する問題を **QBF 問題**と呼び，単に QBF と略記する．

定理 5.13 QBF は **PSPACE** 完全である**．

証明 まず，QBF $\in$ **PSPACE** であることを証明する．ϕ を自由変数を含まない QBF とするとき，図 5.13 に示した関数 eval によって ϕ の値を計算する．ただし，$\phi = \cdots$ は，ϕ が右辺のような形をしていることを表わしている．$\phi'(x)$ は x が ϕ' の自由変数であることを表わし，$a \in \{0,1\}$ に対し，$\phi'(a)$ は $\phi'(x)$ の中のすべての自由変数 x に a を代入して得られるブール式 (の値) を表わす．

eval(ϕ) が呼び出されたときに局所的に使う領域量は，ϕ を保持するために必要な $O(|\phi|)$ だけである．再帰呼出しは多くても，入力論理式が含んでいる $\neg, \wedge, \vee$ の個数回しか起こらないから，系 1.1 より，eval(ϕ) の領域量は $O(|\phi|^2) = O(n^2)$ である．

次に，QBF が **PSPACE** 困難であることを示す．Cook の定理 (定理 5.4) と同様に証明しようとすると，M のステップ数が入力の長さ n の指数関数 c^n になりうるので，M の動作を記述する論理式 (QBF) の長さが多項式で収まらない (したがって，多項式時間で構成できない)．そこで，Savitch の定理 (定理 4.2) の証明と同じ手法を用いて，QBF の長さを $\log_2 c^n$ に押さえるようにする．$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ を $p(n)$ 領域限定の片方向無限単テープ DTM とする．$p(n)$ は具体的に分かっている多項式であり，多項式時間で構成可能である．

$$x \in L(M) \iff \phi_x = 1$$

を満たす QBF ϕ_x を x から構成する多項式時間アルゴリズムを示せばよい．

定理 5.4 の証明と同様に，M の ID とは長さが $p(n)+1$ の，$\Gamma^* Q \Gamma^*$ の語であると考える ($Q \cap \Gamma = \emptyset$)．

$x = x_1 \cdots x_n$ ($x_i \in \Sigma$) とする．ブール変数

$$X_j^I, X_{ja}^I \quad (I \in \mathbf{N},\ 1 \le j \le p(n)+1,\ a \in \Gamma \cup Q)$$

を考える．X_{ja}^I は M の ID I の左から j 番目の記号が a であることを表わす (X_{ja}^I の添字 I は，ID I を表わす整数である)：

$$I \text{ の左から } j \text{ 番目の記号が } a \iff X_{ja}^I = 1.$$

**L.J.Stockmeyer and A.R.Meyer, Word problems requiring exponential time, *Proc. 5th ACM Symp. on the Theory of Comput.*, pp.1-9, 1973.

```
function eval(φ)
begin
    if φ = 1 or φ = 0 then return φ end;
    if φ = φ1 ∨ φ2 then return eval(φ1) ∨ eval(φ2) end;
    if φ = φ1 ∧ φ2 then return eval(φ1) ∧ eval(φ2) end;
    if φ = ¬φ' then return ¬eval(φ') end;
    if φ = ∀xφ'(x) then
        φ'0 ← φ'(0); φ'1 ← φ'(1);
        return φ'0 ∧ φ'1
    end;
    if φ = ∃xφ'(x) then
        φ'0 ← φ'(0); φ'1 ← φ'(1);
        return φ'0 ∨ φ'1
    end
end
```

図 **5.13** QBF ϕ の値を計算する

X_j^I は，ID K の左から j 番目の記号が a である (すなわち，$X_{ja}^K = 1$ である) とき，I の左から j 番目の記号が a であることを $X_j^I \leftrightarrow X_{ja}^K$ によって表わすために使われる (ここに，$x \leftrightarrow y$ は $(\neg x \vee y) \wedge (\neg y \vee x)$ の省略形である):

$$(X_j^I \leftrightarrow X_{ja}^K) = 1 \Longleftrightarrow X_j^I \text{ と } X_{ja}^K \text{ が同じ論理値をとる.}$$

以下で，次の条件を満足するような QBF move(i, I, J) を多項式時間で構成するアルゴリズムを示す:

$$\mathrm{move}(i, I, J) = 1 \Longleftrightarrow I \mathrel{\vdash_M^{(2^i)}} J.$$

ここに，I, J は M の ID で，$I \vdash_M^{(2^i)} J$ は M が I から J へ 2^i ステップ以内で遷移することを表わす．以下で構成する QBF において，1つの ID (例えば I とする) は $p(n)+1$ 個のブール変数の組

$$\langle X_1^I, \ldots, X_{p(n)+1}^I \rangle \text{ あるいは}$$
$$\langle X_{1a_1}^I, \ldots, X_{p(n)+1, a_{p(n)+1}}^I \rangle \quad (\text{ただし}, a_1, \ldots, a_{p(n)+1} \in \Gamma \cup Q)$$

として表わされ，move$(\cdot,\cdot,\cdot)$ を構成する過程において (move$(\cdot,\cdot,\cdot)$ は以下で示すように再帰的に定義されるが，その計算のある段階の再帰呼び出しにおいて)

$$\exists I[\cdots I \cdots] \quad \text{あるいは} \quad \forall I[\cdots I \cdots]$$

という形で初めて登場するときに生成される．大域的カウンタを1つ用意しておき，このように新しい I が出現してブール変数の組が新たに生成されるたびに，カウンタに1を加え，その値を X_j^I や X_{ja}^I の添字 I として使う．

move(i, I, J) は次のように再帰的に定義する．

$i=0$ のとき，$\mathrm{move}(0,I,J)=\phi_0 \vee \phi_1$ とする．ここに，ϕ_0 は $I=J$ を表わすブール式，ϕ_1 は $I \vdash_M J$ を表わすブール式である．$\mathrm{move}(0,I,J)$ が呼び出されたとき，引数 I,J を表わすブール変数の組はそれぞれすでに生成されているので，それらを例えば $\langle X_1^I,\ldots,X_{p(n)+1}^I\rangle$，$\langle X_1^J,\ldots,X_{p(n)+1}^J\rangle$ とすると，ϕ_0 は

$$(X_1^I \leftrightarrow X_1^J) \wedge \ldots \wedge (X_{p(n)+1}^I \leftrightarrow X_{p(n)+1}^J)$$

と表わすことができる．ϕ_1 の構成は読者への問とする(定理5.4の証明における $E_{(2)}$ にならって構成せよ)．

$i>0$ のとき，

$$\mathrm{move}(i,I,J)=\exists K[\mathrm{move}(i-1,I,K) \wedge \mathrm{move}(i-1,K,J)]$$

であるが，この右辺に従ってQBFを作るとその長さは n の指数関数になってしまう(問：なぜか？)．そこで，次のように考える：

$$\begin{aligned}\mathrm{move}(i,I,J) \quad = \quad & \exists K_1 \forall K_2 \forall K_3 \\ & [\,(K_2=I \wedge K_3=K_1) \vee (K_2=K_1 \wedge K_3=J) \\ & \quad \rightarrow \mathrm{move}(i-1,K_2,K_3)\,].\end{aligned}$$

ここに，$\alpha \rightarrow \beta$ は $\neg\alpha \vee \beta$ の省略形である(これらは論理的に等しい)．右辺の [ ] 内は，「$K_2=I$ かつ $K_3=K_1$」または「$K_2=K_1$ かつ $K_3=J$」ならば $\mathrm{move}(i-1,K_2,K_3)$ が成り立つことを表わしている．したがって，$\mathrm{move}(i,I,J)$ の右辺は $\mathrm{move}(i-1,I,K_1)$ かつ $\mathrm{move}(i-1,K_1,J)$ が成り立つような K_1 が存在することを表わしており，それは $\mathrm{move}(i,I,J)$ の意味そのものである．ところで，上式の右辺は次のようなQBFの省略形である．例えば I が $\langle X_{1a_1}^I,\ldots,X_{p(n)+1,a_{p(n)+1}}^I\rangle$ で表わされていた場合($\langle X_1^I,\ldots,X_{p(n)+1}^I\rangle$ の場合も同様)，前半部分 $\exists K_1 \forall K_2 \forall K_3\,[[(K_2=I \wedge K_3=K_1) \vee (\cdots)] \rightarrow \cdots]$ は

$$\begin{aligned}&\exists X_1^{K_1}\cdots\exists X_{p(n)+1}^{K_1}\ \forall X_1^{K_2}\cdots\forall X_{p(n)+1}^{K_2}\ \forall X_1^{K_3}\cdots\forall X_{p(n)+1}^{K_3}\\ &\Bigg[\Bigg[\Big((X_1^{K_2} \leftrightarrow X_{1a_1}^I) \wedge \ldots \wedge (X_{p(n)+1}^{K_2} \leftrightarrow X_{p(n)+1,a_{p(n)+1}}^I)\Big) \wedge \\ &\qquad \Big((X_1^{K_3} \leftrightarrow X_1^{K_1}) \wedge \ldots \wedge (X_{p(n)+1}^{K_3} \leftrightarrow X_{p(n)+1}^{K_1})\Big) \vee \Big(\cdots\Big)\Bigg] \rightarrow \cdots\Bigg]\end{aligned}$$

に置き換える．$\Big(\cdots\Big)$ の部分も同様．残りの部分は，$\mathrm{move}(i-1,K_2,K_3)$ を呼び出して再帰的に作る．

こうして定義した $\mathrm{move}(i,I,J)$ を用いて，求める ϕ_x を次のように定義する．M は $p(n)$ 領域限定であるから，定理4.1より，$c^{p(n)}$ 時間限定であるとしてよい($c>0$ は M に依存して定まる定数)：

$$\phi_x=\exists I_0 \exists I_f\,[\mathrm{move}(\lceil \log_2 c \rceil p(n),I_0,I_f) \wedge \mathrm{Init}(I_0) \wedge \mathrm{Final}(I_f)]\,.$$

ここに，$\mathrm{Init}(I_0)$, $\mathrm{Final}(I_f)$ はそれぞれ I_0, I_f が初期 ID，受理 ID であることを表わすブール式である:

$$\exists I_0 = \exists X^{I_0}_{1q_0} \exists X^{I_0}_{2x_1} \cdots \exists X^{I_0}_{n+1,x_n} \exists X^{I_0}_{n+2,\flat} \cdots \exists X^{I_0}_{p(n)+1,\flat},$$

$$\mathrm{Init}(I_0) = X^{I_0}_{1q_0} \wedge X^{I_0}_{2x_1} \wedge \cdots \wedge X^{I_0}_{n+1,x_n} \bigwedge\nolimits_{n+2 \le i \le p(n)+1} X^{I_0}_{i\flat}.$$

$$\exists I_f = \exists X^{I_f}_1 \cdots \exists X^{I_f}_{p(n)+1},$$

$$\mathrm{Final}(I_f) = \bigvee\nolimits_{\substack{1 \le i \le p(n) \\ q \in F}} \exists X^{I_{iq}}_{iq} \Bigg[\Big((X^{I_f}_i \leftrightarrow X^{I_{iq}}_{iq}) \wedge X^{I_{iq}}_{iq} \Big) \bigwedge\nolimits_{\substack{1 \le j \le p(n) \\ j \ne i}} \bigvee\nolimits_{a \in \Gamma} \exists X^{I_{iq}}_{ja} \Bigg[\Big((X^{I_f}_j \leftrightarrow X^{I_{iq}}_{ja}) \wedge X^{I_{iq}}_{ja} \Big) \wedge \Big(\bigwedge\nolimits_{\substack{b \in \Gamma \cup Q \\ a \ne b}} \neg X^{I_{iq}}_{jb} \Big) \Bigg] \Bigg].$$

最初に $\mathrm{move}(\lceil \log_2 c \rceil p(n), I_0, I_f)$ を呼び出すときに，生成済みのブール変数の組の個数を数えるためのカウンタを 0 に初期化する.

$\mathrm{move}(i, \cdot, \cdot)$ を構成するのにかかる時間を $T(i)$ とすると，$i < \lceil \log_2 c \rceil p(n)$ ならば $\mathrm{move}(i, I, J)$ の右辺の，$\mathrm{move}(i-1, K_2, K_3)$ を除いた部分はたかだか $O(p(n))$ 個しかブール変数を含んでおらず ($\mathrm{move}(\lceil \log_2 c \rceil p(n), I_0, I_f)$ は例外で，$\mathrm{Final}(I_f)$ の部分が $O(p(n)^2)$ 個のブール変数含む)，それらの 1 つを表わすには $O(\log p(n))$ ビットあれば十分であるから,

$$\begin{cases} T(0) & = & O(p(n)) \\ T(i) & = & T(i-1) + O(p(n)^2 \log p(n)) \end{cases}$$

が成り立つ. この漸化式を解くと $T(i) = O(i \cdot p(n)^2 \log p(n))$ であるから，ϕ_x は $T(\lceil \log_2 c \rceil p(n)) = p(n)^{O(1)}$ 時間で構成することができる.

$\mathrm{move}(i, I, J) = 1 \iff I \vdash^{(2^i)}_M J$ の証明は定理 5.4 と同様であるが，次のことに注意する. $\mathrm{move}(i, I, J)$ の構成においては $I = J$ を $\bigwedge_i (X^I_i \leftrightarrow X^J_i)$ で表わし，$\mathrm{Final}(I_f)$ の構成においてはそれに相当すること ($I_f = I_{iq}$) を $\bigwedge_i \big((X^{I_f}_i \leftrightarrow X^{I_{iq}}_i) \wedge X^{I_{iq}}_i \big)$ で表わしている理由は，一般に，x, y をブール変数とするとき,

- $((x \leftrightarrow y) \wedge y) = 1 \iff x = y$ かつ $y = 1$;
- $y = 1$ という前提の下では，$(x \leftrightarrow y) = 1 \iff x = y$;

が成り立つことによる. □

その他の PSPACE 完全問題

(1) **QBF ゲーム** $\phi = Q_1 x_1 Q_2 x_2 \cdots Q_k x_k \psi$ を QBF とする. ただし，i が奇数のとき $Q_i = \exists$，i が偶数のとき $Q_i = \forall$ であり，ψ は $\exists, \forall$ を含まない CNF

の論理式である．このような QBF が与えられたとき，2 人のプレーヤー E と A が交互に $x_1, x_2, \ldots$ の値を決めてゆく (E は ∃ に対応するブール変数の値を決め，A は ∀ に対応するブール変数の値を決める) ゲームを考える．$x_1, \ldots, x_k$ すべての値が決まったら，その値を ψ に代入して得られる値が 1 なら E の勝ち，0 なら A の勝ちである．ϕ が与えられたとき，A がどのような値を選ぼうとも E が勝つことができる (E に**先手必勝法** (winning strategy) が存在する) か否かを判定せよ．この問題は SAT に他ならず，したがって **PSPACE** 完全である．

(2) **尻取りゲーム**　使える単語の集合 W と最初の単語 $s \in W$ を与え，2 人で尻取りゲーム (一度使った単語は二度と使えないものとし，自分の番のときに言うべき単語がなくなった方を負けとする) を行なうとき，先手必勝法が存在するか？

(3) **文脈依存言語の所属問題**　文脈依存文法 G と語 w が与えられたとき，$w \in L(G)$ か否かを判定せよ．文脈依存言語のクラスは **PSPACE** の中でもっともやさしい問題のクラス (**NSPACE**(n)) に等しいことを考えると，これは驚くべき事実である (実は，4.3 節の詰め込み論法が使えるからである)．

(4) **正則表現の等価性判定問題**　2 つの正則表現が等しいか否かを判定せよ (次節参照)．

(5) **石置きゲーム**　1 人石置きゲーム問題および無閉路グラフ上での 2 人石置きゲーム問題は **PSPACE** 完全である (次節参照)．

5.4.2 手に負えない問題

P が実際上計算可能な問題のクラスと考えられるのに対して，**DEXP** あるいはそれより上のクラスに属する問題は，理論上は計算可能であるが，実質的に計算不能と考えてもよい問題である．ここでは，そういった問題の例を示す．

ゲームの先手必勝法

碁や将棋を $n \times n$ の盤上で行なうように一般化したものを考える．任意の局面が与えられたとき，この局面からの先手必勝法 (後手がいかなる手を打とうとも先手が勝つことができるような手の打ち方) が存在するか否かを判定する問題は **DEXP** 完全である．同様に，チェスやチェッカーを $n \times n$ 盤に一般化したゲームにおける先手必勝法の存否を問う問題も **DEXP** 完全である[‡]．碁や

[‡] A.S.Fraenkel and D.Lichtenstein, Computing a perfect strategy for $n \times n$ chess requires time exponential in n, *J. Combinatorial Theory* Ser. A 31, pp.199-214, 1981.
安達博行·亀川裕之·岩田茂樹: $n \times n$ 盤面上の将棋の指数時間完全性について，電子情報通信学会論文誌 J70-D, pp.1843-1852, 1987.
その他の文献についてはこれらの論文を参照せよ．

チェスなどは，実際は 19×19 あるいは 8×8 の盤上で行なうゲームなので異なった局面が有限個しかなく，したがって先手必勝法が存在するかどうかを判定できるはずである (人知を越えてはいるが理論的には可能である) が，上述の結果は，これらがゲームとして成り立つに足るむずかしさを本質的に備えていることを示すものであるといえよう．

これら以外にも，次のような，ゲームの必勝法に関する結果が知られている[§]．

(**石置きゲーム**) 有向グラフ $G=(V,E)$ と $R \subseteq V^3$, $S \subseteq V$, $t \in V$ が与えられる．次のようなゲームを考える．最初，S の頂点に 1 つずつ石が置いてある．R は “規則” の集合で，$(x,y,z) \in R$ $(x \neq y, y \neq z, z \neq x)$ は，頂点 x と頂点 y に石があり，頂点 z に石がないなら，x にある石を z に移すことができることを意味する．1 人でゲームを行なう場合，初期局面 S から R に従って石を移動させ，最終的に頂点 t に石を置くことができるとき，G は**解を持つ**という．また，2 人でゲームを行なう場合，交互に石を動かし，最初に t に石を置いた方を勝ちとする．もし一方の動きがとれなくなった場合は，動けなくなった方の負けとする．**1 人石置きゲーム問題**とは，与えられた G が解を持つかどうかを判定する問題のことである．**2 人石置きゲーム問題**とは，与えられた G に対し，2 人でゲームをしたとき先手必勝かどうかを判定する問題のことである．また，石の個数を $|S|=k$ に固定したものを k **石置きゲーム**という．

2 人石置きゲーム問題は **DEXP** 完全である．1 人石置きゲーム問題および無閉路グラフ上での 2 人石置きゲーム問題は **PSPACE** 完全である．また，無閉路グラフ上での 1 人石置きゲーム問題は **NP** 完全である．2 人 k 石置きゲームは **P** 完全，1 人 k 石置きゲームは **NL** 完全である．

(**ネズミとねこの問題**) G を有向グラフとする．G の頂点には何匹かのねずみがいる (1 つの頂点にたかだか 1 匹)．また，ある頂点にはねこが 1 匹おり，ある頂点にはチーズが 1 つ置いてある．ねずみとねこは辺に沿って頂点から頂点へ移動できる．まず最初にねこが 1 つの辺に沿って動き，次いでねずみのうちの 1 匹が 1 つの辺に沿って動く．以後，ねこと，ねずみのうちの 1 匹が交互に動く．ただし，ねずみは，すでにねずみがいる頂点には移動できない．ある頂点でねこがどれかのねずみと出会ったらねこの勝ち，ねずみのうちのどれか 1 匹がチーズのある頂点に到達したらねずみの勝ちとする．ただし，ねずみがチーズのある頂点に到達したとき，ねこがそこにいてもねずみの勝ちとする．ねこが動けなくなるか，ねずみのすべてが動けなくなったら，動けなくなった方の負けとする．ねこが先手必勝法を持つかどうかを決定する問題は **DEXP** 完全である．

[§]石置きゲーム，および，ねずみとねこの問題については，笠井琢美,「計算量の理論」, 近代科学社, 1987 に詳しい証明がある．

正則表現¶

Σ を有限アルファベットとする．Σ 上の**正則表現** (**正規表現**ともいう，regular expression) と，正則表現 R が表す言語 $\|R\|$ を次のように再帰的に定義する:

1. $\emptyset$, ε はそれぞれ正則表現であり，$\|\emptyset\| = \emptyset$, $\|\varepsilon\| = \{\varepsilon\}$ である．
2. $a \in \Sigma$ なら a は正則表現であり，$\|a\| = \{a\}$ である．
3. R, S がそれぞれ正則表現なら，$(R+S)$, $(R \cdot S)$, (R^*) はいずれも正則表現であり，$\|(R+S)\| = \|R\| \cup \|S\|$, $\|(R \cdot S)\| = \|R\| \cdot \|S\|$, $\|(R^*)\| = \|R\|^*$ である．誤解の生じない範囲で ( ) や $\cdot$ は省略してもよい．
4. 1~3 で定義されるものだけが Σ 上の正則表現である．

例 5.3 (1) $\Sigma = \{a, b\}$ とする．a, $a+b$, $ab+ba$, $(a+b)^*$, $a+b^*$ はそれぞれ Σ 上の正則表現で，$\|a\| = \{a\}$, $\|a+b\| = \{a, b\}$, $\|ab+ba\| = \{ab, ba\}$, $\|(a+b)^*\| = \{a, b\}^*$, $\|a + b^*\| = \{a\} \cup \{b\}^*$ である．

(2) $(\varepsilon + -)(0 + 1(0 + 1)^*0)$ は $\{0, 1, -\}$ 上の正則表現で，偶数の 2 進数の全体を表わす (負の数も含む)．　□

命題 5.3 正則表現が表わす言語は正則集合であり，逆に，任意の正則集合は正則表現で表わすことができる．すなわち，$\mathbf{REG} = \{\|R\| \mid R \text{ は正則表現}\}$.

正則表現の演算子として $+$, $\cdot$, * 以外に，次のような意味を持つ，**べき乗演算子** $\uparrow$ を付け加えることにする: R が正規表現で k が非負整数のとき

$$R \uparrow k = \overbrace{R \cdot R \cdot \ldots \cdot R}^{k}.$$

このような演算子の導入によって，普通の正則表現で表わすと長さが nc^n (c は定数) にもなるような言語を，長さが n という短い正則表現で表わすことができるようになる．それが次のことが成り立つ大きな理由である．

正則表現の等価性判定問題　べき乗演算子を含む 2 つの正則表現 R, S が同じ言語を表わしている ($\|R| = \|S\|$) か否かを判定する問題は **EXPSPACE** 完全である．ここに，$\mathbf{EXPSPACE} = \bigcup_{i \geq 1} \mathbf{DSPACE}(2^{n^i})$ である．たとえ $\|R\| \subseteq \{0,1\}^*$ または $\|S\| = \{0,1\}^*$ であるとしても **EXPSPACE** 完全である．

べき乗演算子を含まない正則表現の等価性判定問題は **PSPACE** 完全である．* 演算子を含まない正則表現の等価性判定問題は co-**NP** 完全である．

¶この項目に関する結果については，巻末参考書案内に挙げた J.E.Hopcroft – J.D.Ullman (和訳あり) の 13 章および C.H.Papadimitriou の 20 章を参照せよ．

さらに，正則表現に演算子 $\neg$ を追加する: R が Σ 上の正則表現のとき，$\|\neg R\| = \Sigma^* - \|R\|$ と定義する．このように拡張された正則表現の等価性判定問題を $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$ 時間で解くアルゴリズムは存在しない．

正確な定義は省略するが，**1 階の形式的理論** (first-order theory) とは，定義域 (例えば $\mathbf{N}$ など．この定義域は特定の定数 (例えば 0 や 1) を含む) とその上の演算 (例えば $+$ や $*$) や述語 (例えば $=$ や $<$) および，演算や述語の意味を定義するための公理を与えることによって定まるものである．1 つの形式的理論 $\mathcal{T}$ が与えられると，$\mathcal{T}$ において正しい命題 (これは，定数を表わす記号，演算や述語を表わす記号，変数を表わす記号，論理記号 $\neg, \wedge, \vee, \forall, \exists$ や括弧 (,) からなる記号列である) の集合として言語 $L(\mathcal{T})$ が定まる．例えば，ゲーデルの不完全性定理 (2.1 節参照) は，自然数の体系を表わす形式的理論 $\mathcal{N} = (\mathbf{N}, +, ^*, =, <, 0, 1)$ が定める言語が帰納的言語でないという主張 (このことを $\mathcal{N}$ は決定可能でないという) に他ならない．

加算だけが許された実数の体系を表わす形式的理論 $(\mathbf{R}, +, =, <, 0, 1)$ の定める言語は決定可能ではあるが，ある定数 $c > 0$ に対して $\mathbf{NTIME}(2^{cn})$ 困難であることが知られている．

5.5 決定問題以外の問題

次のような問題を考える[‖]．

1. 有向グラフ G において頂点 s から頂点 t への道が存在するか？ (GAP)
2. 有向グラフ G において頂点 s から頂点 t への異なる道の個数を求めよ． (#GAP)
3. 辺に重みが付いている完全グラフ G において，重みの和が最小のハミルトン閉路を求めよ．
4. 自然数 m と n の最大公約数を求めよ．

これらの問題のどれもアルゴリズムが関係する問題であるが，要求されている答の形が異なる．問題 1 は決定問題である．すでに 2.3 節で見たように，決定問題は TM による言語の認識問題として定式化することができた．これに対し，実際的な問題の多くは問題 2 ～ 4 のように，入力に対して単に yes/no だけではない値を出力する (すなわち，ある種の関数を計算する) ことを要求する問題である．このような問題を一般に**関数問題** (function problem) という．と

[‖] この節と演習問題 18 については，巻末参考書に掲げた Papadimitriou の 18 章を参照のこと．

くに，問題 2 のように，個数を求める問題は**数え上げ問題** (counting problem) とも呼ばれる．

本書ではこれまですべての問題をむりやり決定問題に直して考えてきた．一般に，本来は決定問題でない問題を "決定問題" という特殊な形にすることによって簡単になる．例えば，問題 2 を解く $\log n$ 領域限定アルゴリズムが存在するなら問題 1 を解く $\log n$ 限定のアルゴリズムはその特別の場合として得られるが，逆は真ではない (例えば，グラフの頂点数が n のとき，道の個数は n の指数関数になりうるので，それらすべてを出力するために指数関数的時間がかかり，$\log n$ 領域限定のアルゴリズムでこれを行なうことはできない)．しかし，すべての決定問題が関数問題より簡単であるというわけでもない．例えば，SAT の関数問題版の 1 つとして次のような問題を考えよう:

FSAT : ブール式 ϕ が与えられたとき，ϕ が充足可能なら ϕ の値を 1 とするような付値 (変数への値の割り当て方のこと．複数あっても，そのうちのどれか適当な 1 つ) を出力し，充足不能なら "no" を出力せよ．

$\phi = \phi(x_1, \ldots, x_n)$ とする．もし，SAT を解く多項式時間アルゴリズムが存在したとすると，次のようにして FSAT を多項式時間で解くことができる．ϕ が充足不能の場合は ok. 充足可能の場合，ϕ の中のすべての x_1 を 0 あるいは 1 で置き換えたブール式をそれぞれ ϕ_0，ϕ_1 とすると，少なくともどちらかが充足可能である．どちらが充足可能であるかに応じて $x_1 = 0$ または $x_1 = 1$ とし，その x_1 の値を ϕ に代入したブール式を考える．今度は，そのブール式において x_2 を考える．という具合に，SAT を解く多項式時間アルゴリズムを $2n$ 回使うと ϕ を充足させる付値が求められる．

さて，決定問題と関数問題の間には次のような深い関係があることを注意しておきたい．$L \in \mathbf{NP}$ とすると，定理 5.2 より，L を多項式時間で検証するアルゴリズム M が存在し，$x \in L \iff M(x,y) = \mathbf{true}$ を満たす y が存在する，が成り立つ (5.2.1 節参照)．そこで，次のような問題 FL を L に**付随する関数問題**と呼ぶ: x が与えられたとき，$M(x,y) = \mathbf{true}$ なる y が存在するならそのうちの適当な 1 つを出力し，そうでなかったら "no" を出力せよ．

$$\mathbf{FNP} = \{\mathrm{F}L \mid L \in \mathbf{NP}\},\quad \mathbf{FP} = \{\mathrm{F}L \mid L \in \mathbf{P}\}$$

と定義する．

命題 5.4 $\mathbf{FP} = \mathbf{FNP}$ である必要十分条件は $\mathbf{P} = \mathbf{NP}$ である．

数え上げ問題

次の問題を考えよう．

#SAT : ブール式 ϕ が与えられたとき，ϕ を充足させるような付値が何通りあるか求めよ．

明らかに，#SAT が解ければ同じアルゴリズムで SAT も解ける．#SAT と同様に，前節で述べた #GAP や，与えられたグラフのハミルトンサイクルの個数を求める問題 (#HAMILTON) などが定義できる (# ○ ○ という命名は，数学では有限集合 A の元の個数を $\#A$ で表わす習慣があることに由来する)．

L を言語 (ある問題のコード) とし，L を多項式時間で検証するアルゴリズム M が存在するとする．$x \in L \Longleftrightarrow M(x, y) = \mathbf{true}$ を満たす y の個数を求める問題を，問題 L の**数え上げ版**という．x は問題 L のインスタンスであり，y の個数 (y を x の**解**と呼ぶ) は 2 進数 ($\{0,1\}^*$ の語) で表わすものとする．このような数え上げ問題のクラスを **#P** で表わす (sharp P とか number P と読む) [†]．#SAT, #HAMILTON, #GAP などは #P の元である．

数え上げ問題についても，**P** 完全問題などと同様に "#**P** 完全問題" を定義することができる．A, B を数え上げ問題 (を表わす関係 $M(x, y)$) とする．Σ^* から Δ^* への関数 f が存在して，

(i) f は対数領域で計算可能，

(i) x が A のインスタンスなら $f(x)$ は B のインスタンス，

(ii) x の解の個数と $f(x)$ の解の個数が等しい

が成り立つとき，A は B に**#還元**されるという．$A_0 \in \#P$ かつ，任意の $A \in$ **#P** が A_0 に#還元されるとき，A_0 を **#P 完全問題** (#**P**-complete problem) という．

$A = (a_{ij})$ を成分が 0 または 1 であるような $n \times n$ 行列とする．

$$\sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}$$

を A の**パーマネント** (permanent) という．ここで，和 $\sum_\pi$ は $1, \cdots, n$ の置換 π すべてについて取る．置換の偶奇によって符号を $+, -$ としたものが A の行列式であることに注意する．パーマネントを計算する問題を PERMANENT で表わす．

命題 5.5 次の問題はいずれも#**P** 完全である:

(a) #SAT (b) #HAMILTON (c) PERMANENT

#P は次のような NTM によって定義することもできる．**カウント TM** (counting Turing machine: **CTM**) とは，入力 x が与えられたとき，x に対する受理計算の個数が $\#\mathrm{accept}_M(x)$ 個であるような NTM のことである．CTM は関数 $x \mapsto \#\mathrm{accept}_M(x)$ を計算していると見ることができる．多項式時間限定の CTM によって計算される関数のクラスを **#P** と定義する．

---

† #P は L.G.Valiant, The complexity of computing the permanent, *Theor. Comput. Sci.* 8, pp.189-201, 1979 で導入された．

演習問題 5

1 アルファベット Σ が1個の記号だけのとき，Σ 上の Post の対応問題は決定可能であることを示せ．

2 2つの文脈自由文法 G_1, G_2 に対して $L(G_1) \cap L(G_2) = \emptyset$ かどうかを決定する問題は計算不能であることを次の2通りの方法で示せ．

(1) 任意の PCP P に対し，$L(G_1) \cap L(G_2) = \{wcw^R \mid w$ は P の解が表わす語$\}$ となる文脈自由文法 G_1, G_2 を作る．

(2) 任意の帰納的可算言語 L に対し，$L = h(L_1 \cap L_2)$ となる文脈自由言語 L_1, L_2 と準同型写像 h が存在する (構成的に作ることができる)‡ ことを使う．

3 $A \le_{poly} B$ かつ $B \le_{poly} A$ であるとき $A \equiv_{poly} B$ と定義すると $\equiv_{poly}$ は同値関係である．$\equiv_{poly}$ の同値類を**多項式時間** m **級** (plynomial-time m-degree) という．次のことを示せ．

(1) 多項式時間 m 級の全体は順序 $\le_{poly}$ のもとで半順序集合である．

(2) $\mathbf{P}$ は $\le_{poly}$ のもとで最小元である．極大な m 級は存在しない．

4 この問題では**神託付きチューリングマシン** (oracle Turing machine: **OTM**) について述べる．A を Σ 上の言語とする．A を**オラクル**とする OTM とは次の条件を満たす多テープ DTM M のことである:

1) M のテープアルファベットは Σ を含む．

2) M のテープのうちの1本は**質問テープ** (query tape) と呼ばれる特別なテープである．

3) M は3つの特別な状態 $q_?$，q_{yes}，q_{no} を持つ．$q_?$ を質問状態 (query state) といい，q_{yes} と q_{no} を回答状態という．

M の動作は普通の DTM のそれとまったく同じであるが，質問状態に入った場合だけ特別な動作を行なう．すなわち，状態 $q_?$ に入ると M は質問テープの内容 (y とする) を調べ，$y \in A$ だったら状態を q_{yes} に変え，$y \notin A$ だったら q_{no} に変える．

M が受理する言語を $L(M^A)$ で表わす．また，$\mathcal{C}$ を計算量のクラス，A をオラクルとするとき，$\mathcal{C}^A = \{L(M^A) \mid L(M) \in \mathcal{C}\}$ と定義し，さらに，$\mathcal{C}_1, \mathcal{C}_2$ が言語のクラスのとき，$\mathcal{C}_1^{\mathcal{C}_2} = \bigcup_{A \in \mathcal{C}_2} \mathcal{C}_1^A$ と定義する．

A をオラクルとする DTM M が存在して $B = L(M^A)$ であるとき，B は “A において帰納的可算” (recursively enumerable in A) であるという．とくに M が停止性 DTM である場合，B は “A において帰納的” であるとか，B は A に **Turing 還元可能** (Turing reducible, **T-還元可能** (T-reducible) と略す) であるいい，$B \le_T A$ と書く．さらに，M が多項式時間限定 DTM である場合，B は A に**多項式時間 T-還元可能** (plynomial-time Turing reducible) であるといい，$B \le_T^{poly} A$ と書く．多項式時間 T-還元を **Cook 還元**ともいう．次のことを示せ．

(1) $B \le_{poly} A$ ならば $B \le_T^{poly} A$.

(2) 任意の A に対して $\mathbf{P}^A \subseteq \mathbf{NP}^A$.

(3) $A \in \mathbf{P}^A$.

(4) $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$, $\mathbf{NP}^{\mathbf{P}} = \mathbf{NP}$, $\mathbf{P}^{\mathbf{NP}} = \mathbf{NP}$.

(5) $B \le_T^{poly} A$ で $A \in \mathbf{P}$ (または $\mathbf{NP}$) ならば $B \in \mathbf{P}$ (または $\mathbf{NP}$).

---

‡S.Ginsburg, S.A.Greibach and M.A.Harrison, Stack automata and compiling, *J. Assoc. Comput. Mach.* 14, pp.172-201, 1967.

(6) $A \in \mathbf{P}$ ならば任意の B に対して $A \leq_T^{poly} B$ である．したがって，$A \leq_T^{poly} B$ であるが $A \leq_m^{poly} B$ でないような A, B が存在する．

(7) 次の命題[§] の (a) を証明せよ (ヒント: オラクルとして **PSPACE** 完全問題を考えよ)．

命題 5.6 (a) $\mathbf{P}^A = \mathbf{NP}^A$ を満たすオラクル A が存在する．
(b) $\mathbf{P}^B \neq \mathbf{NP}^B$ を満たすオラクル B が存在する．

この命題は次のような意味で $\mathbf{P} = \mathbf{NP}$ 問題のむずかしさを示唆している．$\mathbf{P} = \mathbf{NP}$ を証明するために，**NP** (多項式時間限定 NTM) を **P** (多項式時間限定 DTM) で模倣することができたとしよう．すると，その模倣にオラクル B を付加して拡張すること (このようにオラクル版に拡張することを**相対化** (relativization) という) が自然にできて，おそらく $\mathbf{P}^B = \mathbf{NP}^B$ も成り立つ．しかし，上記命題 (b) より，これはあり得ない．また，$\mathbf{P} \neq \mathbf{NP}$ を証明するために，**P** の上の対角線論法によって $\mathbf{NP} - \mathbf{P} \neq \emptyset$ が証明できたとしよう．すると，その証明を相対化するとおそらくそのまま $\mathbf{NP}^A - \mathbf{P}^A \neq \emptyset$ の証明になるであろう．これも上記命題の (a) に反する．そのほかに，変換補題を使う証明法とかについても同様であろう．というわけで，$\mathbf{P} \neq \mathbf{NP}$ の証明には，模倣・対角線論法・変換補題を使うといった手法以外の新しい証明手法が必要なようである．

これまでに述べたもの以外にも多くの還元可能性が研究されている[¶]．

5 $A \leq_m B$ via M で M が線形時間限定 DTM であるとき $A \leq_{lin} B$ と書くことにする．$\leq_{lin}$ のもとで **P** には完全問題が存在しないことを示せ．

6 (1) ハミルトン閉路問題や巡回セールスマン問題は有向グラフに対してもそれぞれ **NP** 完全であることを示せ．

(2) 与えられたグラフがハミルトン道 (すべての頂点をちょうど 1 回通る道のこと．閉路である必要はない) を持つか否かを判定する問題も **NP** 完全であることを示せ．

(3) 3SAT は **NP** 完全であり，2SAT は **P** に属すが，両者の中間に位置する次の問題は **NP** 完全であることを示せ:

MAX2SAT: 2-CNF のブール式 ϕ と正整数 k が与えられたとき，ϕ の節のうち k 個以上を 1 とするような付値が存在するか？

(4) $\neg$ が付いていないリテラル (positive literal) をたかだか 1 個しか含んでいない節を **Horn 節** (Horn clause) という．例えば，$\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4$, $\neg x_1 \vee \neg x_2$, x_1 などは Horn 節である．このタイプのブール式は $x_1 \wedge x_3 \wedge x_4 \rightarrow x_2$, $x_1 \wedge x_2 \rightarrow 0$, $1 \rightarrow x_1$ のように書けることに注目する．最初，すべてのブール変数に 0 を割り当て，ある時点で節 $x_1 \wedge \ldots \wedge x_k \rightarrow y$ の値がまだ 0 のままで，その左辺 ($x_1 \sim x_k$ すべて) が 1 なら y に改めて 1 を割り当てる，ということを値が 0 の節がなくなるまで繰り返すことにより，次の問題が **P** に属すことを示せ．

HORNSAT: ブール式 ϕ がいくつかの Horn 節の論理積であるとき，ϕ が充足可能か否かを判定せよ．

7 公開鍵暗号系において，送り手 A と受け手 B 双方が鍵 e_A, e_B を公開し，それぞれの秘密鍵 d_A, d_B は公開しないものとする．また，両者が使う暗号化の方法 E と復号化の方法 D は共通で，それぞれの秘密鍵 d_A, d_B を使うものとする．A はメッセージ x

---

[§]B.S.Baker, J.Gill and R.Solovay, Relativizations of the P=?NP question, *SIAM J. Comput.* 8, pp.315-332, 1974.

[¶]R.Ladner, N.Lynch and A.Selman, A comparison of polynomial-time reducibilities, *Theor. Comput. Sci.* 1, pp.103-123, 1975.

T.Long, Strong nondeterministic polynomial-time reducibilities, *Theor. Comput. Sci.* 21, pp.1-25, 1982.

と，x を暗号文とみなしてそれを復号化した $D_{d_A}(x)$ とをBに送る．BはAの公開鍵 d_A を使って $D_{d_A}(x)$ を暗号化する:

$$E_{d_A}(D_{d_A}(x)) = D_{d_A}(E_{e_A}(x)) = x.$$

こうして得られた x と，$D_{d_A}(x)$ と一緒に送られてきた x とを比べて一致することを確かめることにより，Bは x が確かにAから送られてきたものであることを確認することができる．これを**デジタル署名** (digital signature) という．RSA システムにおいては上記の等式が成り立つことを示せ．

8 ナップザック問題が **NP** 完全問題であるということに基づく，**ナップザックシステム**と呼ばれる公開鍵暗号系も知られている．

n 個の十分大きい整数 $w_1, \cdots, w_n$ をとり，これを1まとめにして公開鍵 e とする．x を n ビットの2進数とすると，x は $\{1, \cdots, n\}$ の部分集合 X を表わしていると考えることができる．$E_e(x) = \sum_{i \in X} w_i$ を x の暗号文とする．$y = E_e(x)$ から x を復元するために，Bはナップザック問題を解かなければならない．しかし，もしBが互いに素な秘密の整数 N と m を知っており，かつ，各 $w'_i = w_i \cdot m \bmod N$ が $a'_{i+1} > 2a'_i$ を満たすならば，そのようなナップザック問題は容易に解くことができることを示せ．

ナップザックシステムは公開鍵暗号系として使えるのではないかと提案当初は考えられていたが，その後高速の解読法が発見されてしまった‖．これに対し，RSA システムは現在なお高速の解読法が発見されていない．

9 CVP は **P** 完全問題であるが，驚くべきことに，CVP を解く $O(n \log n)$ 時間アルゴリズムが存在することが知られている (n は論理回路のコードの長さ)．この文献†を読んでみよ．

10 素数判定問題 PRIME は **P** に属すか否か分かっていない (この場合，自然数は2進数表現である) が，数を1進数表現した場合には **P** に属すことを示せ．すなわち，$\{1^n \mid n \text{ は素数}\} \in \mathbf{P}$.

11 ナップザック問題を解く $O(nW)$ 時間アルゴリズムを示せ (ヒント: 最初の i 個の品物の中から重さの和が w であるように何個か選んだときの価値の和の最大値を $V(i, w)$ で表わすとき，$V(i, w)$ の漸化式を考えよ)．

このアルゴリズムは入力のサイズ ($n + W$の桁数 $+$ Vの桁数) の多項式ではないが，W の値がたかだか n の多項式である場合には，多項式時間アルゴリズムになっている．そこで，インスタンスの中に現れる数の値が入力サイズ n の多項式で押さえられているような **NP** 完全問題のことを**強 NP 完全** (strongly **NP**-complete) であるという．ほとんどの **NP** 完全問題は強 **NP** 完全である．ナップザック問題，集合分割問題，部分集合和問題，2次合同式の整数解を求める問題など (いずれも5.2節を参照のこと) は強 **NP** 完全ではない．箱詰め問題は，ナップザック問題と同様に数がインスタンスの中に現れるが，強 **NP** 完全であることが知られている．

12 ID 上の遷移を表わす2項関係 $\vdash_M$ が対称律を満たすような NTM M は**対称的** (symmetric) であるという．対数領域限定の対称的 NTM によって受理される言語のクラスを **SL** で表わす．無向グラフにおける到達可能性問題 (1.7節および4.1節参照) を

‖A.Shamir, A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem, *Proc. 23rd IEEE Symp. on Founddations of Comput. Sci.*, pp.142-152, 1982.

†N.J.Pippenger, Fast simulation of combinatorial logic networks by machines without random-access storage, *Proc. 15th Ann. Allerton Conf. on Communication, Control and Computing*, pp.25-33, 1977.

UGAP と呼ぶ．UGAP は **SL** に対して完全であることを示せ[‡]．$\mathbf{SL} \subsetneq \mathbf{NL}$ か否かは未解決である．

13 $n \times n$ 盤上で行なう五目並べを考える (縦・横・斜めのいずれかの方向に自分の石を 5 個連続で最初に並べた方が勝ちである．禁じ手はないものとする)．ある局面が与えられたとき，そこからの先手必勝法が存在するか否かを決定する問題は **PSPACE** に属すことを示せ．

14 任意の自然数 k に対して，言語のクラス $\Sigma_k^p, \Pi_k^p, \Delta_k^p$ をオラクルを用いて次のように定義する (問題 4 参照):

$$\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathbf{P}.$$

$$\Sigma_{k+1}^p = \mathbf{NP}^{\Sigma_k^p},\ \Pi_{k+1}^p = \text{co-}\Sigma_{k+1}^p,\ \Delta_{k+1}^p = \mathbf{P}^{\Sigma_k^p}.$$

$$\mathbf{PH} = \bigcup_{k \geq 0} \Sigma_k^p.$$

$\{\Sigma_k^p\}_{k\geq 0}$ 等の成す階層を**多項式時間階層** (polynomial time hierarchy) という[§]．次のことを示せ．

(1) $\Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

(2) $\Sigma_k^p \subseteq \Pi_k^p$ ならば $\Sigma_k^p = \Pi_k^p$ である．

$\mathbf{PH} \subseteq \mathbf{PSPACE}$ であるが $\mathbf{PH} \subsetneq \mathbf{PSPACE}$ かどうかは未解決である．また，次のいずれも未解決である．

(a) $\Sigma_k^p \subsetneq \Sigma_{k+1}^p$? $(k \geq 0)$

(b) $\Sigma_k^p = \Pi_k^p$? $(k \geq 1)$

(c) $\Delta_{k+1}^p \subsetneq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$? $(k \geq 1)$

15 **ブール階層** (Boolean hierarchy) とは次のように定義される，計算量のクラスの階層 $\{BH_i\}_{i\geq 1}$ のことである:

$$\begin{aligned} &\mathrm{BH}_1 = \mathbf{NP},\quad \mathrm{BH}_2 = \{L_1 \cap L_2^c \mid L_1, L_2 \in \mathbf{NP}\},\\ &\mathrm{BH}_{2i+1} = \{L_1 \cup L_2 \mid L_1 \in \mathrm{BH}_{2i},\ L_2 \in \mathbf{NP}\},\\ &\mathrm{BH}_{2i+2} = \{L_1 \cap L_2 \mid L_1 \in \mathrm{BH}_{2i+1},\ L_2 \in \mathbf{NP}\} \quad (i \geq 1). \end{aligned}$$

$B_i \cup B_{i+1} \subsetneq B_{i+2}$ $(i \geq 1)$ かどうか，今のところ分かっていない．文献によっては，BH_2 は **DP** とか D^p で表わされている．'D' は "difference" の頭文字に由来する $(L_1 \cap L_2^c = L_1 - L_2)$．$\mathrm{BH} = \bigcup_{i\geq 1} \mathrm{BH}_i$ は **NP** のブール閉包である (一般に，$\mathcal{L}$ を言語のクラスとするとき，$\mathcal{L}$ を含み，$\cup, \cap, ^c$ で閉じている最小のクラスを $\mathcal{L}$ の**ブール閉包** (Boolean closure) という) [¶]．

2 つの CNF ブール式 ϕ_1, ϕ_2 が与えられたとき，一方が充足可能でもう一方が充足可能でないことを判定する問題 (**SAT(2)**) は BH_2 完全であることを証明せよ．

16 次の各問題は **FP** に属すことを示せ．

[‡] H.R.Lewis and C.H.Papadimitriou, Symmetric space-bounded computation, *Theor. Comput. Sci.* 19, pp.161-187, 1982.

[§] L.J.Stockmeyer, The polynomial-time hierarchy, *Theor. Comput. Sci.* 3, pp.1-22, 1977.

[¶] J.Cai, Gundermann, J.Hartmanis, L.A.Hemachandra and V.Sewelson, The Boolean hierarchy I: Structural properties, II: Applications, *SIAM J. Comput.* 17, pp.1232-1252, 1988; 18, pp.95-111, 1989.

DP については C.H.Papadimitoriou and M.Yannakakis, The complexity of facets (and some facets of complexity), *J. Comput. Sys. Sci.* 28, pp.244-259, 1984. を見よ．

(a) 2つの自然数の最大公約数を求める
(b) FHORNSAT (FSAT を Horn 節に限定した問題)

17 自然数 n が与えられたとき，その素因数分解 $n = p_1^{k_1}p_2^{k_2}\cdots p_m^{k_m}$，および $p_1, p_2, \ldots, p_m$ が素数であることの確認証 (5.2.1 節参照) を求める問題を**素因数分解問題** (FACTORING) という．FACTORING $\in$ **FNP** であることを示せ．

素因数分解問題はもっとも有名な **FNP** 問題であり，**FP** に属すか否か分かっていない．公開鍵暗号系はこのことに基づいている．しかし，多項式時間アルゴリズムではないものの，非常に高速なアルゴリズムが知られている[‖]．

この問題が非常にむずかしい関数問題であると信じられている理由の一つは，この問題で計算する必要のあるのが "全域関数" であるということにある (この種の問題を **TFNP** (total **FNP**) と呼ぶ) [**]．すなわち，いかなる自然数 n に対しても，その素因数分解が必ず存在し，それを求めなければならない．**TFNP** に属す問題の他の例は，次に述べる **ANOTHER-HAMILTON** である:「グラフ G と，G のハミルトン閉路の1つが与えられたとき，もう1つ別のハミルトン閉路を求めよ.」

(a) どの頂点の次数も3であるグラフを**3次グラフ** (cubic graph) という．任意の3次グラフは偶数個のハミルトン閉路を持つことを示せ．
(b) ANOTHER-HAMILTON は **FNP** 完全であることを示せ．

18 言語のクラス Few**P**, **PP**, **GP**, $\oplus$**P** を次のように定義する[††]:

$L \in$ Few**P** $\stackrel{\text{def}}{\iff}$ 多項式 p が存在して，任意の x に対して $\#\text{accept}_M(x) \le p(x)$，かつ「$x \in L \iff \#\text{accept}_M(x) > 0$」を満たす多項式時間限定 NTM M が存在する．

$L \in$ **PP** $\stackrel{\text{def}}{\iff}$ 「$x \in L \iff \#\text{accept}_M(x) \ge f(x)$」を満たす多項式時間限定 NTM M と $f \in$ **FP** が存在する．

$L \in$ **GP** $\stackrel{\text{def}}{\iff}$ 「$x \in L \iff \#\text{accept}_M(x) = f(x)$」を満たす多項式時間限定 NTM M と $f \in$ **FP** が存在する．

$L \in \oplus$**P** $\stackrel{\text{def}}{\iff}$ 「$x \in L \iff \#\text{accept}_M(x)$ が奇数」を満たす多項式時間限定 NTM M が存在する．

ここに，$\#\text{accept}_M(x)$ は入力 x に対する M の受理計算の個数を表わす．

$\oplus$**P** は補集合で閉じていること ($\oplus$**P** = co-$\oplus$**P**) が知られているので，上記の定義において「奇数個」の代わりに「偶数個」としてもよい．$\oplus$**P** は odd P とか parity P と読む．定義より明らかに，**UP** $\subseteq$ Few**P** $\subseteq$ **NP** である．**NP** $\subseteq$ **PP**, Few**P** $\subseteq \oplus$**P**, **GP** $\subseteq$ **PP** であることが知られている．$\oplus$**P** については，7章の演習問題8も参照せよ．**PP** については 7.1 節で詳述する．

$\oplus$SAT とか $\oplus$**P** 完全問題とかが容易に定義できる．$\oplus$SAT は $\oplus$**P** 完全であることを示せ．

‖ A.K.Lenstra and H.W.Lenstra, Algorithms in number theory, *The Handbook of Theoretical Computer Science, Vol. I: Algorithms and Complexity*, ed. by J. van Leeuwen, MIT Press, 1990.

** N.Megiddo and C.H.Papadimitoriou, On total functions, existence theorems, and computational complexity, *Theor. Comput. Sci.* 81, pp.317-324, 1991. TFNP はこの論文で初めて考察された．

†† C.H.Papadmitoriou ans S.Zachos, Two remarks on the power of counting, *Lecture Note in Comput. Sci.* 145, Springer- Verlag, pp.269-276, 1983.

J.-Y.Cai and L.Hemachandra, On the power of parity polynomial time, *Proc. 6th Ann. Symp. on Theor. Comput. Sci., Lecture Notes in Comput. Sci.* 349, Springer–Verlag, pp.229-239, 1989.

6

並列計算

例えば $a*x+b*y$ の計算をするとき，計算機が1台しかなければ $a*x$ の計算と $b*y$ の計算を同時に行なうことはできないから，$u \leftarrow a*x$, $v \leftarrow b*y$, $u+v$ のようにどうしても3ステップかかってしまうのに対して，計算機が2台あれば $a*x$ の計算と $b*y$ の計算は互いに独立なので同時に並行して行なうことが可能であり，$a*x+b*y$ の計算は2ステップですむ．このように多数のプロセッサを同時に使って行なう計算 (**並列計算**) は，前章までのようにプロセッサが1台しか使えない計算 (**逐次計算**) に比べて計算量をどの程度まで改良できるのであろうか？ 交代性 TM や論理回路など，並列計算のモデルを導入し，このことを考察するのがこの章の主題である．

6.1 交代性チューリングマシン

並列計算の最初のモデルとして，NTM を次のように拡張する*．**kテープ交代性チューリングマシン** (k-tape alternating Turing machine, 以後 **ATM** と略記する) とは5つ組 $M=(Q,\Sigma,\Gamma,\delta,q_0,g)$ のことをいう．ここに，$Q,\Sigma,\Gamma,\delta,q_0$ はそれぞれ状態の集合，入力アルファベット，テープアルファベット，遷移関数 (動作関数)，初期状態を表わし，NTM のそれと同じものである．

$$g: Q \rightarrow \{\forall, \exists, \mathbf{accept}, \mathbf{reject}\}$$

は状態を分類する関数である．状態 $q \in Q$ は，$g(q)=\forall$ のとき**全称状態** (universal state)，$g(q)=\exists$ のとき**存在状態** (existential state)，$g(q)=\mathbf{accept}$ のとき**受理状態** (accepting state)，$g(q)=\mathbf{reject}$ のとき**拒否状態** (rejecting state) と呼ばれる．

---

*A.K.Chandra, D.Kozen and L.J.Stockmeyer, Alternation, *J. Assoc. Comput. Mach.* 28, pp.114-133, 1981.

NTM の場合と同様に,

$$I = (q, (\alpha_1, h_1), \ldots, (\alpha_k, h_k)) \in Q \times (\Gamma^* \times \mathbf{N})^k$$

の元 (とくに $k = 1$ の場合には $\Gamma^* Q \Gamma^*$ の元で表わすこともある) のことを M の時点表示 (ID) という. q はそのときの状態, α_i は第 i テープの内容, h_i は第 i テープヘッドの位置 (α_i の左端からの距離) を表わす. q が全称/存在/受理/拒否状態のときの I をそれぞれ全称/存在/ 受理/拒否 ID という. g の適用を ID にまで拡張し, $g(q, (\alpha_1, h_1), \ldots, (\alpha_k, h_k)) = g(q)$ で定義する. ID から ID への直接推移を表わす 2 項関係 $\vdash_M$, およびその反射推移閉包 $\vdash_M^*$ の定義は NTM のそれと同じである.

ATM が行なう "計算" を次のように定義する. 各頂点に ID がラベル付けされた有限の根付き木 T が次の条件を満足するならば, T を入力 x に対する M の**計算木** (computation tree) という:

1. 根のラベルは入力 x に対する M の初期 ID $(q_0, (x, 1), (\flat, 1), \ldots, (\flat, 1))$ である.
2. T の任意の内点 (葉以外の頂点のこと) を π とし, そのラベルを I とする. π は全称 ID または存在 ID であり, $\{J \mid I \vdash_M J\} = \{J_1, \ldots, J_n\}$ とすると π は n 個の子を持ちそれらのラベルは $J_1, \ldots, J_n$ である ($J_1, \ldots, J_n$ の順序はどうでもよい: 図 6.1 参照).

以後, π が計算木の頂点のとき, π に付けられたラベルをしばしば π 自身で表わす. 例えば, $g(\pi)$ は $g(\pi$のラベル$)$ のことであるし, 全称頂点とは $g(\pi) = \forall$ であるような頂点 π のこと, 存在頂点とは $g(\pi) = \exists$ であるような頂点 π のことである. また, $I \vdash_M^* J$ は, 計算木上で頂点 I から頂点 J へ至るの道が存在することを表わしている.

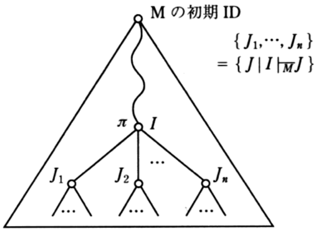


図 6.1 ATM M の計算木

T を計算木とする. T に 0 または 1 を対応させる関数 ℓ を次のように再帰的に定義する:

T が頂点 π 1つだけから成るとき，

$$\ell(\pi) = \begin{cases} 1 & g(\pi) = \textbf{accept} \text{ のとき} \\ 0 & \text{その他}. \end{cases}$$

T の根が π で，その子が $\sigma_1, \ldots, \sigma_k$ のとき，

$$\ell\left(\begin{array}{c} \pi \\ \sigma_1 \quad \cdots \quad \sigma_k \\ \triangle T_1 \quad \triangle T_k \end{array}\right) = \begin{cases} 1 & g(\pi) = \forall \text{ かつ} \\ & \ell\left(\overset{\sigma_1}{\triangle T_1}\right) = \cdots = \ell\left(\overset{\sigma_k}{\triangle T_k}\right) = 1 \text{ のとき} \\ 1 & g(\pi) = \exists \text{ かつ} \\ & \ell\left(\overset{\sigma_1}{\triangle T_1}\right), \ \ldots, \ \ell\left(\overset{\sigma_k}{\triangle T_k}\right) \text{ の中の} \\ & \text{少なくとも 1 つが 1 のとき} \\ 0 & \text{その他}. \end{cases}$$

すでに述べた計算木の定義より，T の内点のラベルはすべて全称 ID か存在 ID である．また，上述の定義より，入力 x に対する計算木は一意的に定まらないことに注意する (問 6.1 (1) 参照)．$\ell(T) = 1$ であるとき，T を**受理計算木** (accepting computation tree) という．入力 x に対する M の受理計算木が存在するとき M は x を**受理**する．NTM の場合と同様に，M が受理する語の集合を $L(M)$ で表わす．

以上の定義より，ATM は受理状態または拒否状態に入ったら必ず停止するものと考えてよい．

問 6.1 上述の定義について以下のことを考察せよ．

(1) 計算木の定義における条件 1, 2 それぞれを，根付き木 $T_{M,x}$ を作るための手続きの初期ステップ，展開ステップと考える (x に対する M の計算が停止しなければ $T_{M,x}$ は有限ではない)．$T_{M,x}$ をきちんと定義し，すでに述べたすべての定義をそれに基づいたものに改めよ．入力 x に対する M の計算木はどれも，この $T_{M,x}$ の (根を共有する) 部分木である．

(2) 上述の定義では，受理/拒否状態が全称/存在状態を兼ねることが許されておらず，また，受理/拒否状態からはどこへも遷移が許されていない (もっと正確に言うと，たとえ受理/拒否状態に対して δ が定義されていても，そのような遷移を含む計算木が許されていない)．このような制約がないように上述の定義を改め，両者に本質的差がないことを示せ．

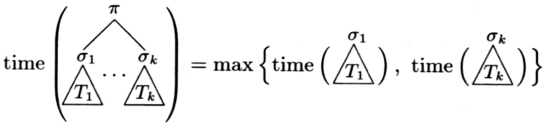
次の定理の証明は読者への演習問題とする．

定理 6.1 任意の ATM M に対し，$L(M)$ は帰納的可算言語である．

さて次に，ATM の時間量，領域量を定義しよう．M を ATM とし，$t(n), s(n)$ を $\mathbf{N}$ から $\mathbf{N}$ への関数とする．任意の入力 x に対して，「$x \in L(M)$ ならば，x

に対する M の受理計算木 T で $\text{heigt}(T) \le t(|x|)$ であるものが存在する」が成り立つとき，M は $t(n)$ **時間限定**であるという．また，任意の入力 x に対して，「$x \in L(M)$ ならば，どの頂点における ID もたかだか $s(n)$ 領域しか使っていないような M の計算木が存在する」が成り立つとき，M は $s(n)$ **領域限定**であるという．

この定義の意味について注釈を加えておこう．この定義では計算木をもとに，時間量 time が，π が全称頂点の場合には



のように定義されている．このことは，，...，の計算が並行して行なわれていることを意味する．すなわち，π から遷移する際に ATM のコピーが k 個作られて，以後それぞれのコピーが独立に $\sigma_1, \ldots, \sigma_k$ 以降の計算を並列に進める (これが，ATM が並列計算のモデルとされる所以である)．π が存在頂点の場合には，上式の右辺は $\max\{\cdots\}$ ではなく $\min\{\cdots\}$ であり，そのことは，，...，のうちのどれか 1 つが受理 ID に到達すればよい (前ページの受理の定義において「$x \in L(M)$ なら… 存在する」としているのはこのことである) が，これは非決定性の遷移にならない．これに対し，π が全称頂点の場合には，すべてが受理 ID に到達することが求められる．

全称状態から動作 ●●● を開始することを「∀ に ●●● を行なう (universal に ●●● する)」という．これに対し，存在状態から一連の動作 ○○○ を開始することを「∃ に ○○○ を行なう (existential に ○○○ する)」という．すなわち，∃ な動作とは推測にもとづく非決定性の動作のことであり，∀ な動作とは並列に行なう動作のことである．

$t(n)$ 時間限定 ATM が受理する言語のクラスを $\mathbf{ATIME}(t(n))$ で，$s(n)$ 領域時間限定 ATM が受理する言語のクラスを $\mathbf{ASPACE}(s(n))$ で表わす．また，$t(n)$ 時間限定かつ $s(n)$ 領域限定である ATM が受理する言語のクラスを $\mathbf{ATISP}(t(n), s(n))$ で表わす．さらに，

$$\mathbf{AL} = \bigcup_{i \ge 1} \mathbf{ASPACE}(\log n), \quad \mathbf{AP} = \bigcup_{i \ge 1} \mathbf{ATIME}(n^i)$$
$$\mathbf{APSPACE} = \bigcup_{i \ge 1} \mathbf{ASPACE}(n^i), \quad \mathbf{AEXP} = \bigcup_{i \ge 1} \mathbf{ATIME}(2^{n^i})$$

と定義する．

例 6.1 (1) $L_1, L_2 \in \mathbf{ATISP}(t(n), s(n))$ ならば $L_1 \cap L_2 \in \mathbf{ATISP}(t(n), s(n))$ であることを示そう．

L_i を受理する ATM を $M_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, s_{0i}, g_i)$ とする $(i = 1, 2)$．$Q_1 \cap Q_2 = \emptyset$ と仮定してもよい．$Q = Q_1 \cup Q_2 \cup \{s_0\}$，$\Sigma = \Sigma_1 \cup \Sigma_2$，$\Gamma = \Gamma_1 \cup \Gamma_2$，$g(s_0) = \forall$，$q \in Q_i$ なら $g(q) = g_i(q)$ とし，任意の $a \in \Sigma \cup \{\flat\}$ に対し $\delta(s_0, a) = \{(s_{01}, a, 0), (s_{02}, a, 0)\}$，$q \in Q_i$ かつ $a \in \Sigma_i$ なら $\delta(q, a) = \delta_i(q, a)$ と定義する．$M = (Q, \Sigma, \Gamma, \delta, s_0, g)$ は $L_1 \cap L_2$ を受理する．

(2) $L = \{x\#y\#x \mid x, y \in \Sigma^*, \ \# \notin \Sigma\}$ とする．L を受理する $O(n)$ 時間限定の単テープ ATM が存在することを示そう．この言語は $O(n)$ 時間限定の単テープ DTM では受理できない (定理 3.9 の証明参照)．

入力 w が与えられたとき次の (a)～(c) をチェックする $O(n)$ 時間限定の ATM M を構成する．$|w| = n$ とする．

(a) w は $x\#y\#z$ の形をしている．

(b) x は z の接頭語である．

(c) z は x の接頭語である．

<u>ステップ 1</u>　M は (a) をまずチェックする．これは $O(n)$ 時間でできる．

<u>ステップ 2</u>　(b) と (c) のチェックは以下に述べる方法で行なう．(b) についてだけ述べるが，(c) も同様である．

<u>ステップ 2.1</u>　M はテープを 4 トラックに分割し，第 1 トラックに入力 w を書き込む (図 6.2 参照)．これは $O(n)$ 時間でできる．

<u>ステップ 2.2</u>　次に x の各ます目 i $(1 \leq i \leq \ell_x = |x|)$ を $\forall$ に選び，その第 2 トラックに印を付ける．

<u>ステップ 2.3</u>　それぞれの i ごとに z の j 番目のます目 $(1 \leq j \leq \ell_z = |z|)$ を $\exists$ に選び，その第 2 トラックに印を付ける．(b) をチェックするためには $\forall i \exists j [x_i = z_j \wedge i = j]$ が成り立つこと，すなわち次のことをチェックすればよい:

(b1) マークを付けた 2 つの文字が等しい $(x_i = z_j)$．

(b2) $i = j$.

<u>ステップ 2.3.1</u>　(b1) のチェックは $O(n)$ 時間でできる．

<u>ステップ 2.3.2</u>　(b2) のチェックのために以下のことを行なう．

<u>ステップ 2.3.2.1</u>　第 3 トラック上に 2 進数 $m_1, m_2, \ldots, m_t \in \{0,1\}^*$ $(t < n)$ を下記の ① ～③ が成り立つように $\exists$ に推測して書き込む ($O(n)$ 時間でできる)．ただし，m_1 は x の左端から始まり，m_t は z の右端から始まること．

① $|m_i| = O(\log n)$.

② m_i の左端と m_{i+1} の左端との距離は $O(n/\log n)$.

③ m_i の左端はテープの左端から $\overline{m_i}$ 番目のます目である．ここに，$\overline{m_i}$ は

m_i の値を表わす．$\overline{m_1} = 1$, $\overline{m_t} = |x\#y\#z|$ でなければならない．

条件 ①,② は M の時間量を $O(n)$ に押さえるために必要な条件であり，そのような m_i が存在しさえすればよいので，とくにチェックする必要はない (なぜなら，$t(n)$ 時間限定の ATM によって受理されるとは，高さが $t(n)$ 以下の受理計算木が存在することであるから) が，③ が成り立っていることはチェックする必要がある．以下のステップでこのチェックを行なう．

ステップ 2.3.2.2　各 i $(2 \leq i \leq t)$ に対して

$$\overline{m_i} - \overline{m_{i-1}} = 「m_i\text{の左端と } m_{i-1}\text{の左端との距離}」$$

であるか否かを ∀ にチェックする．そのためには，距離 $O(n/\log n)$ だけ離れた 2 ビットの引き算をたかだか $|m_i| = O(\log n)$ 回やればよいので，それにかかる時間は $O(n)$ である (作業用に第 4 トラックを使う)．条件が満たされなかった場合は入力を拒否する．生成された m_i の値によっては $\omega(n)$ 時間かかってしまうこともあるが，条件 ①,② が満たされている場合 (そのような場合は必ず存在する) には $O(n)$ 時間でできる．

ステップ 2.3.2.3　x_i の位置 i と z_j の位置 j を計算する．そのためには，x_i, z_j それぞれにもっとも近い $m_k, m_{k'}$ を見つけて，$\overline{m_k}$ +「x_i と m_kの左端との距離」と $\overline{m_{k'}}$ +「z_j と $m_{k'}$の左端との距離」を計算すればよい (計算結果を第 2 トラックに書き込む) が，これは $O(n)$ 時間でできる．

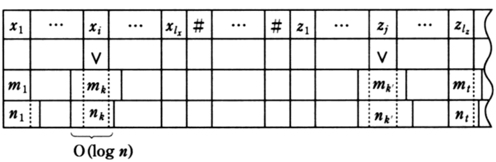


図 6.2　M の 4 つのトラック

ステップ 2.3.2.4　$i = j$ をチェックするための準備として，各 m_i と左端をそろえて 2 進数 $n_1, \ldots, n_t$ を第 4 トラックにオーバーラップしないように ∃ に書き込む．これにかかる時間は $O(n)$ である．

ステップ 2.3.2.5　$i = 2, \ldots, t$ について ∀ に $n_i = n_{i-1}$ かどうかをチェックする．そうでなかったら入力を拒否する．そうだったら，

$$i = 「x_i\text{の近辺の } n_k」,\quad j = 「z_j\text{の近辺の } n_{k'}」$$

かどうかを調べ，正しかったら入力を受理，正しくなかったら拒否する．これにかかる時間は $O(n)$ である．　□

例 6.1 (2) で用いた "一里塚技法" (milestones method) を用いて，次の命題を証明することができる[†]．3.2.2 節の結果と比較せよ．

命題 6.1 任意の $t(n)$ 時間限定 k テープ ATM M に対して，M と同じ言語を受理する $t(n)$ 時間限定の単テープ ATM が存在する．

また，線形加速定理，線形領域圧縮定理は ATM に対しても成り立つ．DTM に対する階層定理 (定理 4.6，定理 4.7) と同様に，ATM に対して次の階層定理が成り立つ．

定理 6.2 $t_1(n) \geq t_2(n)$ で $t_1(n)$ は時間構成可能関数とする．$\inf_{n\to\infty} t_2(n)/t_1(n) = 0$ ならば $\mathbf{ASPACE}(t_2(n)) \subsetneq \mathbf{ASPACE}(t_1(n))$ である．

定理 6.3 $s_1(n) \geq s_2(n) \geq \log n$ で $s_1(n), s_2(n)$ は領域構成可能関数とする．$\inf_{n\to\infty} s_2(n)/s_1(n) = 0$ ならば $\mathbf{ASPACE}(s_2(n)) \subsetneq \mathbf{ASPACE}(s_1(n))$ である．

証明 定理 6.2 だけを証明しよう．定理 6.3 の証明も同様である．

命題 6.1 により，単テープ ATM だけを考えれば十分であるから，$x \in \{0,1\}^*$ をコードとする単テープ ATM を M_x で表わす．任意の i に対し，コード x と $0^i x$ は同じ ATM を表わすものとする．入力 $x \in \Sigma^*$ に対して次のように動作する 2 テープ ATM M を考える．$|x| = n$ とする．

1. はじめに M は第 2 テープ上の $t_1(n)$ 領域に印を付け，第 2 テープのヘッドをその左端に置く．$t_1(n)$ は時間構成可能だから，これは $O(t_1(n))$ 時間でできる．以後，この領域は時間のカウンタとして用いる．すなわち，M が 1 ステップ動作するごとに第 2 テープのヘッドを 1 こま右に動かす (カウンタから 1 減じることに相当する)．
2. 次いで，M は x を入力として第 1 テープを使い M_x を模倣する．ただし，受理/拒否状態および存在/全称状態を M_x とは逆にして模倣を行なう．したがって，M_x が入力 x を拒否 (受理) したら M は x を受理 (拒否) する．
3. M_x が途中で受理/拒否状態に入ることなくカウンタが 0 になったら (すなわち，第 2 テープヘッドが $\flat$ を読んだら) 入力を受理する．

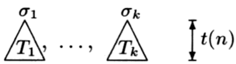
明らかに M は $t_1(n)$ 時間限定であるから $L(M) \in \mathbf{ATIME}(t_1(n))$ である．命題 6.1 より，時間量を変えることなく M を単テープ ATM とすることができる．

[†] W.J.Paul, E.J.Prauß and R.Reischuk, On alternation, *Acta Inform.* 14, pp.243-255, 391-403, 1980.

$L(M) \in \mathbf{ATIME}(t_2(n))$ と仮定すると，$L(M)$ を受理する $t_2(n)$ 時間限定単テープ ATM が存在する．それを M_{x_0} とする．$\inf_{n\to\infty} t_2(n)/t_1(n) = 0$ だから十分大きい i を選ぶと $t_2(|0^i x_0|) < t_1(|0^i x_0|)$ とできる．この $0^i x_0$ をあらためて x_0 と考える．M_{x_0} に x_0 を入力したとき計算は $t_1(|x_0|)$ ステップ以内に終了するから，M の動作の定義より $x_0 \in L(M) \Longleftrightarrow x_0 \notin L(M_{x_0}) = L(M)$ であり，これは矛盾である． □

系 6.1 $\mathbf{ATIME}(t(n))$ は補集合で閉じている．

上述の例や定理で見たように，ATM の能力は DTM より確かに向上している．ATM の時間量の定義によれば，DTM なら逐次 k 回の分岐を行なって計算しなければならない部分



(その場合にかかる時間は $k^{t(n)}$) が並列計算によって $t(n)$ 時間でできてしまうので，$t(n)$ 時間限定 ATM は $k^{t(n)}$ 時間限定の DTM と同等以上の能力を持つと考えるのは自然である．このことは $\mathbf{ATIME}(t(n)) \supseteq \mathbf{DTIME}(k^{t(n)}) \supseteq \mathbf{DSPACE}(t(n)^c)$ を示唆している ($c > 0$ は定数)．事実，次のことが成り立つ:

補題 6.1 $s(n) \geq n$ ならば $\mathbf{NSPACE}(s(n)) \subseteq \mathbf{ATIME}(s(n)^2)$ である．

証明 M を $s(n)$ 領域限定 NTM とし，x を長さ n の入力とする．x に対する M の異なる ID はたかだか $c^{s(n)}$ 個しかない (c は正定数) から，高さ $2^{d\cdot s(n)}$ 以下の計算木だけを考えればよい ($d = \lceil \log_2 c \rceil$)．次の ATM M' を考える:

1. 入力 x に対する M の初期 ID $\mathrm{Init}_{M,x}$ をテープ上に書く．
2. M の受理 ID I_f を ∃ に推測してテープ上に書く．
3. 次の関数 reachable($\mathrm{Init}_{M,x}, I_f$) を実行し，その値が **true** なら x を受理し，**false** なら拒否する (reachable$(\cdot,\cdot)$ は，受理/拒否する代わりに関数値 **true**/**false** を返す ATM である．ATM を認識装置として定義する場合には，行 3.4 は書かなくてもよいものである: 問 6.3 参照)．

```
function reachable(I1, I2)
begin
    3.1. if I1 = I2 or I1 ⊢_M I2 then return true;
    3.2. M の ID I3 を ∃ に推測する;
    3.3. reachable(I1, I3) と reachable(I3, I2) を ∀ に実行する;
    3.4. if reachable(I1, I3) = true  and reachable(I3, I2) = true
             then return true else return false
         end
end
```

任意の ID I, J に対して，reachable$(I, J) = \mathbf{true} \iff I \vdash_M^* J$ であることが帰納法で容易に証明でき，

$$\begin{aligned} x \in L(M) &\iff \mathrm{Init}_{M,x} \vdash_M^* I_f \text{ となる受理 ID } I_f \text{ が存在する} \\ &\iff \mathrm{reachable}(\mathrm{Init}_{M,x}, I_f) = \mathbf{true} \text{ となる } I_f \text{ が存在する} \\ &\iff x \in L(M') \end{aligned}$$

である．

M' の時間量について考えよう．I, J を M の ID とするとき，2^i ステップ以下で I から J へ遷移する ($I \vdash_M^k J$, $k \leq 2^i$) ならば

$$I \overset{(i)}{\Longrightarrow} J$$

と書くことにする．$x \in L(M)$ なら $\mathrm{Init}_{M,x} \overset{2^{d \cdot s(n)}}{\Longrightarrow} I_f$ となる受理 ID I_f が存在する．このような I_f に対し，$\mathrm{Init}_{M,x} \overset{2^{d \cdot s(n)-1}}{\Longrightarrow} I_3 \overset{2^{d \cdot s(n)-1}}{\Longrightarrow} I_f$ となる I_3 が必ず存在する．以後，関数 reachable(I_1, I_2) が再帰的に呼び出されるたびに行 3.2 においていつもこのように I_1 と I_2 のちょうど中間の I_3 が ∃ に選ばれれば，再帰はたかだか $d \cdot s(n)$ 回で終了する (そのような場合が必ず存在する)．一方，reachable の行 3.1〜3.4 を 1 回実行するのにかかる時間は再帰呼び出しの部分を除けば $O(s(n))$ であるから，行 3 の実行にかかる時間は $d \cdot s(n) O(s(n)) = O(s(n)^2)$ である．よって，M' が使う時間は総計で $O(s(n)^2)$ である． □

一方，補題 6.1 と逆の包含関係については，次のことが成り立つ．

補題 6.2 $t(n) \geq n$ かつ $t(n)$ は領域構成可能とすると，$\mathbf{ATIME}(t(n)) \subseteq \mathbf{DSPACE}(t(n)^2)$ である．

証明 M を $t(n)$ 時間限定 ATM とする．$x \in L(M)$, $|x| = n$ とする．x に対する M の計算木 T_x で高さが $t(n)$ 以下であるものが存在する．M を模倣

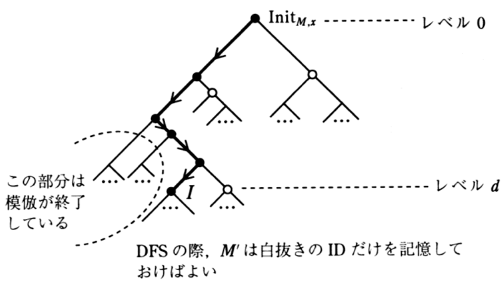


図 6.3 T_x を DFS する

```
function acceptable(I, d)
begin
   1. if d ≥ t(n) then
         if I が受理 ID then return true else return false end
      end;
   2. if I が存在 ID then
         I の子を I_1, ..., I_k とする;
         for i ← 1 to k do
              if acceptable(I_i, d + 1) then return true end;
              acceptabel(I_i, d + 1) の計算に使った領域を消去する
              /* acceptable(I_{i+1}, d + 1) の計算において同じ領域を使う */
         end;
         return false
      end;
   3. if I が全称 ID then
         I の子を I_1, ..., I_k とする;
         alltrue ← true;
         for i ← 1 to k do
              alltrue ← alltrue and acceptable(I_i, d + 1);
              acceptabel(I_i, d + 1) の計算に使った領域を消去する
         end;
         if alltrue then return true else return false end
      end;
   4. /* その他の場合 (I からの遷移が不可能のとき) */ return false
end
```

図 6.4 T_x の部分木 $T_{I,d}$ を計算する関数

する DTM M' を以下で述べるように構成する．M' は T_x を DFS でたどるように M を模倣する (図 6.3 参照).

まず，M' は M の初期 ID $\mathrm{Init}_{M,x}$ をテープ上に書き，別のテープの $t(n)$ 領域に印を付ける．これらのことは $t(n)$ 領域を使うだけでできる．$t(n)$ は図 6.4 の関数 acceptable の計算において使う．次に，M' は関数 acceptable($\mathrm{Init}_{M,x}$, 0) を計算し，その関数値が **true** なら x を受理し，**false** なら拒否する．図 6.4 の acceptable(I, d) は，T_x 上の深さ d の位置にある頂点 (I はその頂点の ID である) を根とする T_x の部分木 ($T_{I,d}$ とする) が受理計算木 (すなわち，$\ell(T_{I,d}) = 1$) であるか否かを判定する関数である.

M のどの ID の長さも $O(t(n))$ であるから，関数 acceptable が局所的に使う領域量は $O(t(n))$ である．acceptable の再帰の深さはたかだか $t(n)$ であるから，系 1.1 より，M の領域量は $O(t(n)^2)$ である． □

問 6.2 補題 6.2 を，問 3.2 の方法にもとづいて定理 4.1 (1) と同様に証明しようとしてもうまくゆかない．なぜか？

補題 6.3 $s(n) \geq \log n$ は $d^{s(n)}$ 時間で領域構成可能な関数であるとする．このとき，$\mathbf{ASPACE}(s(n)) \subseteq \bigcup_{c>0} \mathbf{DTIME}(c^{s(n)})$ である．

証明 M を $s(n)$ 領域限定 ATM とし，x を M への入力とする ($|x| = n$)．M を模倣する DTM M' を以下のように構成する．M' は，入力 x に対する M の計算木を葉の側から根へ向かって作っていくように M の計算を模倣することによって $x \in L(M)$ かどうかを判定する (図 6.5)．入力 x に対する M の初期 ID を $\mathrm{Init}_{M,x}$ とする．

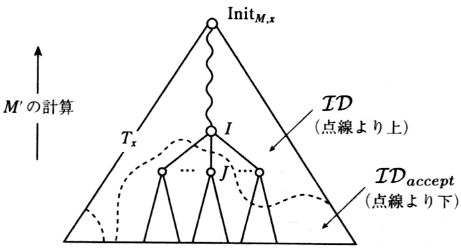


図 6.5 M' は T_x を下から上へ模倣する

1. M' は第 1 テープの $s(n)$ 領域 ($s(n)$ 個の連続するます目) に印を付ける．仮定から，これに要する時間は $O(d^{s(n)})$ である．
2. M' は，M の ID のうち $s(n)$ 領域しか使っていないものをすべて求めて第 2 テープに書き出す．この ID の集合を $\mathcal{ID}$ で表わす．$\mathcal{ID}$ を生成するために，第 1 テープ上に生成した $s(n)$ を使う．このような ID の総数はたかだか $b^{s(n)}$ (b は正定数) であり，その生成にかかる時間は $O(b^{s(n)})$ である (問：なぜか？)．したがって，第 2 テープ上で $\mathcal{ID}$ が占める領域の大きさ $|\mathcal{ID}|$ は $O(s(n)b^{s(n)})$ である．$s(n) \geq \log n$ だから $O(s(n)b^{s(n)}) = O(b^{s(n)})$ である．
3. 第 2 テープ上の ID のうち，受理 ID すべてに印を付ける．印を付けられた ID の集合を $\mathcal{ID}_{accept}$ で表わす．この作業にかかる時間は $O(|\mathcal{ID}|)$ である．
4. $\mathcal{ID} \leftarrow \mathcal{ID} - \mathcal{ID}_{accept}$，updated ← "yes" とする．以下の操作で，$\mathcal{ID}$ の元のうち受理 ID になることがわかったものを順次除去していく．変数 updated の値は，そのような除去対象 ID が見つかったとき updated = "yes" であり，そうでないとき (すなわち，$\mathcal{ID}$ の中にはもはや受理 ID がないとわかったとき) updated = "no" である．これにかかる時間は $O(|\mathcal{ID}|)$ である．

5. $|\mathcal{ID}| \neq \emptyset$ かつ updated = "yes" の間，次の 5.1～5.2 を反復実行する．
   5.1. updated ← "no";
   5.2. **forall** $I \in \mathcal{ID}$ **do**
   if I が存在 ID で $I \vdash_M J$ となる $J \in \mathcal{ID}_{accept}$ が存在する
   or I が全称 ID で $I \vdash_M J$ となるすべての J に対して $J \in \mathcal{ID}_{accept}$
   then
   $\mathcal{ID}_{accept} \leftarrow \mathcal{ID}_{accept} \cup \{I\}$;
   $\mathcal{ID} \leftarrow \mathcal{ID} - \{I\}$;
   updated ← "yes"
   end
   end
6. $\mathrm{Init}_{M,x} \in \mathcal{ID}_{accept}$ なら x を受理し，そうでないなら拒否する．
   $\mathrm{Init}_{M,x} \in \mathcal{ID}_{accept}$ は $x \in L(M)$ を意味する．

$L(M') = L(M)$ であることは明らかであろう．ループ 5 が反復されている間は updated = "yes" であり，反復 1 回ごとに $\mathcal{ID}$ の元は 1 つずつ減少するからループ 5 はたかだか $O(b^{s(n)})$ 回実行される．ループ 5.2 の反復回数が $O(|\mathcal{ID}|)$ であり，ループ 5.2 の本体を 1 回実行するのにかかる時間は $O(|\mathcal{ID}|)$ であるから，ループ 5 の実行にかかる時間は $O(b^{s(n)}|\mathcal{ID}|^2) = (c^{s(n)})$ である (c は適当な正定数)． □

補題 6.4 $t(n) \geq n$ ならば $\mathbf{DTIME}(t(n)) \subseteq \mathbf{ASPACE}(\log t(n))$ である．

証明 M を $t(n)$ 時間限定 DTM とする．系 3.1 (3.2 節) より，M は $t(n)^2$ 時間限定単テープ DTM と仮定してもよい．さらに，次の仮定をおいても一般性を失わない:

- M の受理状態は 1 つだけである．それを q_f とする．
- M の入力テープは右に片方向無限であり，左端には特殊な記号 # がつねに書いてある (図 6.6)．



図 6.6 M の初期 ID (左) と I_i の j 番目の記号 (右)

M の ID は

$$uqv \quad (u, v \in \Gamma^*,\ q \in Q)$$

で表わすことにする．Q は M の状態集合，Γ は M のテープアルファベットである．入力 x に対する M の初期 ID は $I_0 = q_0 x$ である．

$x \in L(M)$ の計算

$$I_0 \vdash_M I_1 \vdash_M \cdots \vdash_M I_T, \quad T \le t(n)^2$$

の各 I_i $(0 \le i \le T)$ の j 番目の記号は I_{i-1} の $j-1$ 番目，j 番目，$j+1$ 番目の記号だけで定まる．そこで，$\Delta = Q \cup \Gamma \cup \{\#\}$ とし I_i の j 番目の記号を X_{ij} とするとき，X_{ij} を定める関数を step : $\Delta^3 \to \Delta$ とする:

$$X_{ij} = \mathrm{step}(X_{i-1,j-1}, X_{i-1,j}, X_{i-1,j+1}).$$

このとき，

$$x \in L(M) \iff X_{ij} = q_f \text{ となる } 0 \le i \le t(n)^2,\ 1 \le j \le t(n)^2 \text{ が存在する}$$

であることに注意する．M を模倣する ATM M' を次のように作る:

1. i, j を ∃ に選ぶ．
2. 次の関数 check(i, j, q_f) を実行し，その値が **true** なら x を受理し，**false** なら拒否する．

function check(i, j, a)
/* I_i の j 番目の記号が a か否かを調べる */
begin
  2.1. **if** $i = 0$ **then** /* 初期 ID */
      if I_0 の j 番目の記号が a
        then return true else return false
      end
    end;
  2.2. **if** $j = 1$ **then**
      if $a = \#$ **then return true else return false end**
    end;
  2.3. a_{-1}, a_0, a_1 を ∃ に選ぶ;
  2.4. $a' = \mathrm{step}(a_{-1}, a_0, a_1)$ を計算する;
  2.5. **if** $a \ne a'$ **then**
      return false
    else
      各 $k \in \{-1, 0, 1\}$ に対して ∀ に $C_k \leftarrow \mathrm{check}(i-1, j+k, a_k)$ を実行する;
      if C_1 **and** C_2 **and** C_3 **then return true else return false end**
    end
end

i に関する数学的帰納法で，

$$I_i \text{ が状態 } q \text{ を含む} \iff \mathrm{check}(i, j, q) = \mathbf{true} \text{ となる } j \text{ が存在する}$$

であることが容易に証明できる．よって，

$$\begin{aligned} x \in L(M) &\iff q_f \text{ を含む } I_i \text{ が存在する} \\ &\iff \mathrm{check}(i, j, q_f) = \mathbf{true} \text{ となる } i, j \text{ が存在する} \\ &\iff x \in L(M') \end{aligned}$$

が成り立つ．M' が使う領域量は，i, j を記憶しておくために必要な $O(\log t(n)^2) = O(\log t(n))$ だけである． □

問 6.3 次の論述はなぜ必要ないのか (どこが正しくないか)?:
関数 check(·,·,·) は，行 2.5 において再帰呼び出しをする際に局所変数 i, j, a 等をスタックに積む必要がない (なぜなら，再帰呼び出しから戻って来た後にそれらを使うことがないから) ので，M' が使う領域量は ···

補題 6.1 ～ 6.4より，次の定理を得る．

定理 6.4 DTM の階層 **DL** ⊆ **P** ⊆ **PSPACE** ⊆ **DEXP** ⊆ ··· は，ATM の階層においては 1 つずつずれる．すなわち，

$$\begin{array}{ccccccccccc} \mathbf{DL} & \subseteq & \mathbf{P} & \subseteq & \mathbf{PSPACE} & \subseteq & \mathbf{DEXP} & \subseteq & \mathbf{EXPSPACE} & \subseteq & \cdots \\ & & \| & & \| & & \| & & \| & & \\ & & \mathbf{AL} & \subseteq & \mathbf{AP} & \subseteq & \mathbf{APSPACE} & \subseteq & \mathbf{AEXP} & \subseteq & \cdots \end{array}$$

が成り立つ．ここに，$\mathbf{EXPSPACE} = \bigcup_{i \geq 1} \mathbf{DSPACE}(2^{n^i})$ である．

証明 **AL** = **P** と **AP** = **PSPACE** を示そう．他も同様である．

$$\begin{array}{lcll} \mathbf{AL} & = & \mathbf{ASPACE}(\log n) & (\mathbf{AL} \text{ の定義}) \\ & \subseteq & \bigcup_c \mathbf{DTIME}(c^{\log n}) & (\text{補題 } 6.3\,(c \text{ は適当な定数})) \\ & = & \bigcup_d \mathbf{DTIME}(n^d) & (d \text{ は定数}) \\ & = & \mathbf{P} & (\mathbf{P} \text{ の定義}) \\ & \subseteq & \bigcup_i \mathbf{ASPACE}(\log n^i) & (\text{補題 } 6.4) \\ & = & \mathbf{ASPACE}(\log n) & (\mathbf{ASPACE} \text{ に対する線形領域圧縮定理}) \\ & = & \mathbf{AL}. & \end{array}$$

一方，

$$\begin{array}{lcll} \mathbf{AP} & = & \bigcup_i \mathbf{ATIME}(n^i) & (\mathbf{AP} \text{ の定義}) \\ & \subseteq & \bigcup_i \mathbf{DSPACE}(n^{2i}) & (\text{補題 } 6.2) \\ & = & \mathbf{PSPACE} & (\mathbf{PSPACE} \text{ の定義}) \\ & = & \bigcup_i \mathbf{NSPACE}(n^i) & (\text{系 } 4.2 : \text{Savitch の定理}) \\ & \subseteq & \bigcup_i \mathbf{ATIME}(n^{2i}) & (\text{補題 } 6.1) \\ & = & \mathbf{AP}. & \end{array}$$

$\log n$ は $O(n)$ 時間で領域構成可能，n^i は領域構成可能であることに注意する (問: 証明せよ)． □

問 6.4 定理 6.4の証明の残りを行なえ．

6.2 論理回路族

6.2.1 論理回路によって計算可能な関数

関数 $f:\{0,1\}^n \to \{0,1\}^m$ を n 入力 m 出力の**ブール関数**と呼び，これらの全体を $\mathbf{B}_n^m$ で表わす．定数 0 と 1 は 0 変数ブール関数であると考えることができる．

例 6.2 (1) $\neg \in \mathbf{B}_1^1$ である．また，$\wedge, \vee \in \mathbf{B}_2^1$ である．以後，この章では $\neg x$ を $\overline{x}$ で，$x \wedge y$ を $x \cdot y$ あるいは xy で，$x \vee y$ を $x+y$ で表わす．

(2) $x \oplus y = \begin{cases} 0 & x=y \text{ のとき} \\ 1 & x \neq y \text{ のとき} \end{cases}$

で定義されるブール関数 $\oplus \in \mathbf{B}_2^1$ を**排他的論理和** (exclusive or, **XOR**) という．

$$x \oplus y = x\overline{y} + \overline{x}y = (x+y)(\overline{x}+\overline{y})$$

である．また，$\oplus$ は交換律 $(x \oplus y = y \oplus x)$，結合律 $(x \oplus (y \oplus z) = (x \oplus y) \oplus z)$ を満たす．

(3) $x\,|\,y = \overline{xy} = \overline{x} + \overline{y}$ で定義される素子 (ゲート) を **NAND 素子** (not and の意) という．また，$x \downarrow y = \overline{x+y} = \overline{x} \cdot \overline{y}$ で定義される素子を **NOR 素子**という (not or の意)． □

問 6.5 排他的論理和に関して次の性質が成り立つことを示せ．
(a) $x \oplus x \oplus \cdots \oplus x = x \Longleftrightarrow$ 左辺の x の個数が奇数．

(b) $x_1 \oplus x_2 \oplus \cdots \oplus x_n = \begin{cases} 1 & x_1, x_2, \ldots, x_n \text{の中の 1 が奇数個のとき} \\ 0 & x_1, x_2, \ldots, x_n \text{の中の 1 が偶数個のとき.} \end{cases}$

これを n **変数パリティ関数** (parity function) という．

定理 1.4 (1.5 節) より，任意のブール関数は $\neg$ と $\vee$ だけ，あるいは $\neg$ と $\wedge$ だけで表わすことができる．このことと，$\overline{x} = x|x = x \downarrow x$ であることより，任意のブール関数は $|$ だけ，あるいは $\downarrow$ だけでも表わすことができることが導かれる．したがって，論理回路の基底としては $\{\neg, \vee\}$, $\{\neg, \wedge\}$, $\{\,|\,\}$, $\{\downarrow\}$ のいずれでも十分であるが，今後は，基底を

$$\Omega = \{\neg, \wedge, \vee\}$$

に制限した論理回路だけを考える．例 6.2 (2) より，Ω が $\oplus$ を含んでいると考えても以後の議論に影響しない．また，$\neg, \wedge, \vee$ のファンインをそれぞれ 1, 2, 2 に制限する．ファンアウトに制限はない．さらに，出力ノードが y_1 1 つしかないものだけを考える (以後，y_1 を単に y と書くことにする)．このような論理回路 $C(x_1, \ldots, x_n)$ はブール関数

$$\varphi_C : \{0,1\}^n \to \{0,1\}$$

を定める．すなわち，$\varphi_C(a_1,\ldots,a_n)$ は，C の入力変数 $x_1,\ldots,x_n$ に $a_1,\ldots,a_n$ を与えたときの出力変数 y の値である．このとき，C は φ_C を**計算する**という．2つの論理回路 $C(x_1,\ldots,x_n)$ と $C'(x_1,\ldots,x_n)$ は，任意の入力に対して出力する値が等しいとき，**同値**であるという．

補題 6.5 C を，基底が $\Omega=\{\neg,\wedge,\vee\}$ でファンインが無制限の論理回路とする．次の条件を満足する，C と同値な論理回路 C' が存在する:

(1) C' のどのゲートもファンインがたかだか 2 である．

(2) $\mathrm{size}(C') \le 2\cdot\mathrm{size}(C)$.

(3) $\mathrm{depth}(C') \le \mathrm{depth}(C)\cdot\log_2\mathrm{size}(C)$.

証明　図 6.7左に示したファンインが k のゲート g (g は $\wedge(x_1,\ldots,x_k)=x_1\cdots x_n$ または $\vee(x_1,\ldots,x_n)=x_1+\cdots+x_n$ である) は，図 6.7右に示したような，すべてのゲートのファンインが 2 の論理回路 C_g と同値である．図 6.7右の論理回路は，2分木の高さができるだけ低くなるように組んだもので，$\mathrm{depth}(C_g)\le\lceil\log_2 k\rceil$, $\mathrm{size}(C_g)\le 2\cdot\mathrm{size}(g)$ (ただし，size にはファンインの値 k も加えている) が成り立つ．$\cdot$, $+$ はそれぞれ結合律を満たすので g と C_g は同値であり，すべての $\wedge,\vee$ をこのような回路に置き換えた回路を C' とすれば補題の条件を満たす．$k\le\mathrm{size}(C)$ であることに注意する．　□

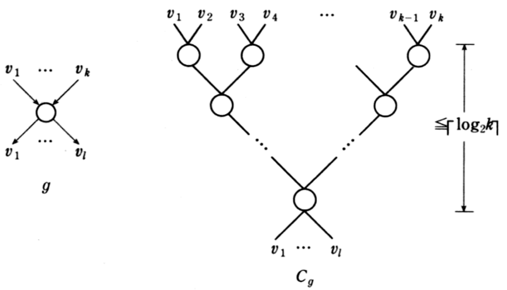


図 6.7　ファンイン k の素子 (左) と同値なファンイン 2 の論理回路 (右)

問 6.6 論理回路 C のサイズや深さをそれぞれ「C の辺の個数」(あるいは「ゲートの個数」),「入力ノードから出力ノードへ至る，長さが最大の道の上にあるゲートの個数」と定義することもある．この場合，補題 6.5をどのように修正したらよいか．

C_n を n 入力 1 出力論理回路とするとき，$\mathbf{C} = \{C_n\}_{n \geq 0}$ を**論理回路族** (family of circuits) という．C_n が n 変数ブール関数 $f_n \in \mathbf{B}_n^1$ を計算するとき (すなわち，$\varphi_{C_n} = f_n$ であるとき)，論理回路族 $\mathbf{C} = \{C_n\}_{n \geq 0}$ は関数族 $\{f_n\}_{n \geq 0}$ を**計算する**という．$z : \mathbf{N} \to \mathbf{N}$, $d : \mathbf{N} \to \mathbf{N}$ とする．$\mathrm{size}(C_n) \leq z(n)$ であるとき $\mathbf{C}$ の**サイズ**は (たかだか) $z(n)$ であるといい，$\mathrm{depth}(C_n) \leq d(n)$ であるとき**深さ**は (たかだか) $d(n)$ であるという．

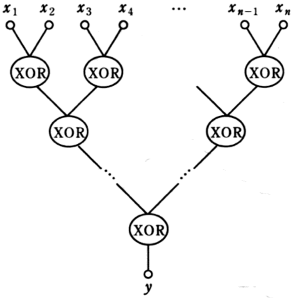


図 **6.8** n 入力パリティ回路

例 6.3 (1) 図 6.8 は，n 変数のパリティ関数を計算する論理回路である．この回路のサイズは $2n$，深さは $\lceil \log_2 n \rceil + 1$ である．

(2) 2 つの n ビット数の加算回路 ($2n$ 入力 $n+1$ 出力) を考えよう:

$$\begin{array}{r} a_n \cdots a_i \ldots a_2 a_1 \\ + \quad b_n \cdots b_i \ \ldots \ b_2 b_1 \\ \hline s_{n+1} s_n \cdots s_i \ \ldots \ s_2 s_1 \end{array}$$

和の i ビット目の値 s_i，および $i+1$ ビット目への桁上がり c_i は

$$\begin{cases} s_i = a_i \oplus b_i \oplus c_{i-1}, \quad s_{n+1} = c_n \\ c_0 = 0, \quad c_i = a_i b_i + (a_i \oplus b_i) c_{i-1} \quad (i = 1, \ldots, n) \end{cases}$$

で与えられる (問: なぜか？)．これを計算する回路を**全加算器** (full adder) という (図 6.9 では FA で表わしている)．n 個の全加算器を図 6.9 のように組み合わせると求める加算回路 (**直列 2 進加算器** (ripple-carry adder) という) が得られる．この回路はサイズも深さも $O(n)$ である． □

問 6.7 (1) 真理値 0,1 を成分とする行列を**ブール行列** (Boolean matrix) という．2 つの $n \times n$ ブール行列 $A = (a_{ij})$, $B = (b_{ij})$ の積 $C = (c_{ij})$ を $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$

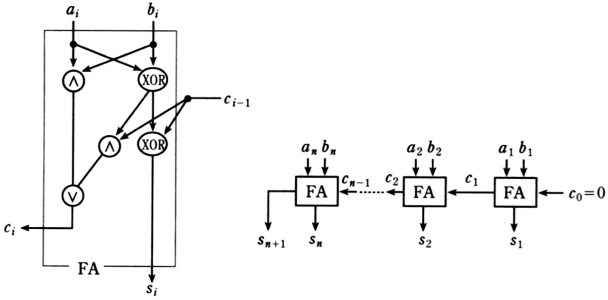


図 6.9 直列 2 進加算器

で定義する．ただし，$\cdot$ は論理積，$\sum$ は論理和である．2 つのブール行列の積を計算する $2n^2$ 入力 n^2 出力の論理回路でサイズが $O(n^3)$ かつ深さが $O(\log n)$ のものを示せ．

(2) $$\mathrm{MAJ}_n(x_1,\ldots,x_n) = \begin{cases} 1 & x_1,\ldots,x_n \text{ の中の 1 の個数} \geq n/2 \text{ のとき} \\ 0 & \text{その他} \end{cases}$$

により定義される関数 MAJ_n を n 変数の**多数決関数** (majority function) という．多数決関数の族を計算する，サイズが $O(n \log n)$ かつ深さが $O(\log^2 n)$ の論理回路族を示せ．

6.2.2 論理回路と言語，クラス NC

これまでは論理回路はブール関数を計算するものとしか考えなかったが，以下で，論理回路をアルファベット $\{0,1\}$ 上の言語に結びつける．これによって，論理回路を決定問題を計算するアルゴリズムと考えることができる (2.3 節で見たように，任意の決定問題は $\{0,1\}$ 上の言語として符号化できることを思い出されたい)．

$\mathbf{C} = \{C_n\}_{n\geq 0}$ を論理回路族とし，$L \subseteq \{0,1\}^*$ とする．$\mathbf{C}$ が L の特性関数族 $\{\chi_n\}_{n\geq 0}$

$$\chi_n(x_1,\ldots,x_n) = \begin{cases} 1 & x_1\cdots x_n \in L \text{ のとき} \\ 0 & x_1\cdots x_n \notin L \text{ のとき} \end{cases}$$

を計算するとき，$\mathbf{C}$ は L を**受理** (accept) するとか，**認識** (recognize) するという．$\mathbf{C}$ が受理する言語を $L(\mathbf{C})$ で表わす:

$$L(\mathbf{C}) = \{x_1\cdots x_n \in \{0,1\}^* \mid C_n(x_1,\ldots,x_n) = 1,\ n \geq 0\}.$$

例 6.4 図 6.10 の論理回路族 $\mathbf{C}_{WW}$ は $\mathrm{WW} = \{ww \mid w \in \{0,1\}^*\}$ を受理する．図 6.10 において，$\ominus$ は入力 x_1, x_2 に対して $(x_1 + \overline{x_2})(\overline{x_1} + x_2)$ を出力

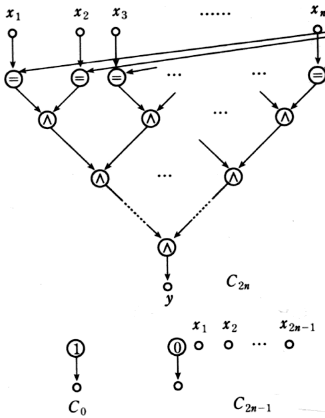


図 **6.10** 言語 WW を受理する論理回路族 $\mathbf{C}_{WW}$

する 2 入力 1 出力の回路である．入力 x_1, x_2 に対する ㊁ の出力が 1 である必要十分条件は $x_1 = x_2$ である．明らかに，$\mathbf{C}_{WW}$ のサイズは $O(n)$，深さは $O(\log n)$ である． □

さて，論理回路を，決定問題を解くための装置として使えることは分かったが，論理回路族の定義には何の制約もなかった (構成的であること，すなわち，論理回路族を構成する方法 (=アルゴリズム) が存在することさえ求められていない) ので，このままでは，アルゴリズム的問題の計算量を調べるための道具として論理回路は使えない．実際，次の定理に述べるように，サイズや深さをきつく制限しても構成的でない論理回路族が存在する．

定理 6.5 帰納的でない言語 (すなわち，計算不能問題) を受理する，多項式サイズの論理回路族が存在する．

証明 $L \subseteq \{0,1\}^*$ を帰納的でない任意の言語とし，tally 言語

$$U = \{1^n \mid n \text{ の 2 進数表現は } L \text{ の元である}\}$$

を考える．U も帰納的でない．なぜなら，L は U へ指数関数時間で m-還元可能だから，U を帰納的とすると L も帰納的となってしまうからである．

次のような論理回路族 $\mathbf{C} = \{C_n\}_{n \geq 0}$ を考える．$1^n \in U$ なる n に対しては，$n-1$ 個の AND ゲートによってすべての入力の論理積をとって出力とする論

理回路を C_n とする．$1^n \notin U$ なる n に対しては，すべての入力ノードの出次数が 0 で，定数ノード 0 から出力ノードへの辺 1 つだけがあるような論理回路を C_n とする．$\mathbf{C}$ が U を受理すること，また，$\mathbf{C}$ が多項式サイズであることは明らかであろう． □

定理 6.5 の証明で示した論理回路 $\mathbf{C}$ は構成的なものでないことに注意する．しかし，論理回路を計算量を調べるための道具として使うためには，論理回路に構成的であるという条件をつけるだけでは不十分である．というのは，対象となる問題の計算量を論理回路の静的な計算量 (サイズと深さ) によって測りたいのであるが，論理回路がたとえ構成的であったとしても論理回路自身の持つ計算能力があまりにも高いと，対象となる問題の計算量が論理回路の計算量のなかに埋没してしまってわからなくなってしまうからである．そこで導入されるのが "一様性" という概念である．一様性の定義にはさまざまなものがあるが，本書では以下に述べる定義を用いる．

はじめに，論理回路の符号化について復習しておこう (5.3.1 節参照)．$C = (V, E)$, $\tau : V \to \Omega \cup \{0, 1, x_1, \ldots, x_n, y_1, \ldots, y_m\}$ を n 入力 1 出力の論理回路とする．$V = \{v_0, \ldots, v_{p-1}\}$ とし，v_0 をただ 1 つの出力ノードとする．p は C のサイズである．$(v_i, v_j) \in E$ のとき，v_i を v_j の**親**，v_j を v_i の**子**という (回路図は，親が子の上側になるように描く)．頂点 v_i のコード $\overline{v_i}$ を $[\overline{i}, type, \overline{j}, \overline{k}]$ と定義する．ここに，$\overline{i}, \overline{j}, \overline{k}$ は $i, j, k \in \mathbf{N}$ の 2 進数表現，$type$ は v_i の "タイプ" であり，

1. v_i が定数ノードのとき $type = \tau(v_i)$, $j = k = 0$.
2. v_i が入力ノードのとき $type =$ "in", $j = k = 0$.
3. v_i が出力ノードのとき $type =$ "out", $(v_j, v_i) \in E$, $k = 0$.
4. v_i が NOT ゲートのとき $type = \tau(v_i) = \neg$, $(v_j, v_i) \in E$, $k = 0$.
5. v_i が AND ゲートまたは OR ゲートのとき $type = \tau(v_i)$, $(v_j, v_i) \in E$, $(v_k, v_i) \in E$.

である．アルファベット $\{0, 1,$ "in", "out", $[,], ,, \neg, \wedge, \vee\}$ 上の語 $\overline{v_0} \cdots \overline{v_{p-1}}$ を C のコード $\overline{C}$ とする．$\overline{C}$ は簡潔な符号化である．

$\mathbf{C} = \{C_n\}_{n \geq 0}$ を論理回路族とする．$n \in \mathbf{N}$ が与えられると C_n を生成する $\log n$ 領域限定のアルゴリズムが存在するとき，すなわち，入力 $1^n \in \{1\}^*$ に対して $\overline{C_n}$ を出力する $O(\log n)$ 領域限定変換機が存在するとき，$\mathbf{C}$ は**対数一様** (log-space uniform) であるという．条件 "$O(\log n)$" を "$O(\log \mathrm{size}(C_n))$" に緩めた場合，$\mathbf{C}$ は **BC 一様**であるという．

サイズが $O(z(n))$，深さが $O(d(n))$ の対数一様または BC 一様な論理回路族が受理する言語のクラスをそれぞれ

$$\mathbf{U}_{log}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(z(n),d(n)),\ \mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(z(n),d(n))$$

で表わす．ときには，これらのクラスを言語のクラスではなくブール関数のクラスと考えることもある (本書でも断りなく両方の意味で用いるので，読者は文脈で判断されたい) [‡]．

補題 6.6 対数一様な論理回路族と多項式サイズの BC 一様な論理回路族とは一致する．また，任意の k に対して次の関係が成り立つ:
$\mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)},\log^k n) = \mathbf{U}_{log}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)},\log^k n)$.

証明 はじめに，$\mathbf{C}=\{C_n\}_{n\geq 0}$ が対数一様な論理回路族ならば $\mathbf{C}$ は多項式サイズ，すなわち $\mathrm{size}(C_n)=n^{O(1)}$ であることを示そう．入力 1^n から $\overline{C_n}$ を生成する $O(\log n)$ 領域限定変換機を M とする．定数 $c>0$ が存在して M は $O(c^{\log n})=n^{O(1)}$ 時間限定である (定理 4.1) から，$|\overline{C_n}|=O(\mathrm{size}(C_n)\log\mathrm{size}(C_n))=n^{O(1)}$ である．よって，$\mathbf{C}$ は多項式サイズである．ゆえに，$\log\mathrm{size}(C_n)=O(\log n)$ である．このことは，$\overline{C_n}$ が $O(\log\mathrm{size}(C_n))$ 領域で生成できることを意味し，$\mathbf{C}$ は BC 一様である．

逆に，$\mathbf{C}=\{C_n\}_{n\geq 0}$ をサイズが $p(n)$ の BC 一様論理回路族 ($p(n)$ は多項式) とすると，定義より，1^n から $\overline{C_n}$ を生成する $O(\log p(n))=O(\log n)$ 領域限定の変換機が存在する．よって，$\mathbf{C}$ は対数一様である． □

この補題を踏まえ，問題のクラス **NC** を次のように定義する[§]．**NC** は実際的なサイズ (= 多項式サイズ) の並列計算装置 (= 論理回路．ゲートをプロセッサと考えるならば実際的な個数のプロセッサ) によって非常に高速に (= 対数多項式時間で) 計算できる問題のクラスと考えることができる．

$$\begin{aligned}\mathbf{NC}^k &= \mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)},\log^k n),\ k=0,1,\ldots\\ \mathbf{NC} &= \textstyle\bigcup_{k\geq 0}\mathbf{NC}^k.\end{aligned}$$

ここに，$\mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)},\log^k n)$ は $\bigcup_{c>0}\mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}$ $(n^c,\log^k n)$ の略記である．

定理 6.6 (1) $\mathbf{NC}\subseteq\mathbf{P}$.　(2) $\mathbf{NC}^k\subseteq\mathbf{DSPACE}(\log^k n)$.

[‡] 一様論理回路の概念を初めて提唱したのは Borodin であり，Cook はそれを拡張した．BC は彼らの名前に因む．対数一様性は Ruzzo による:

A.Borodin, On relating time and space to size and depth, *SIAM J. Comput.* 6, pp.733-744, 1977.

S.A.Cook, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, *Proc. 11th ACM Symp. on Theory of Comput.*, pp.338-245, 1979.

W.L.Ruzzo, On uniform circuit complexity, *J. Comput. Sys. Sci.* 22, pp.365-383, 1981.

[§] NC を初めて提唱したのは N.Pippenger である．NC は "Nick's class" の頭文字をとって S.Cook が命名した:

N.Pippenger, On simultaneous resource bounds, *Proc. 20th IEEE Symp. on Foundations of Comput. Sci.*, pp.307-311, 1979.

証明 (1) $L \in \mathbf{NC}^k$ を受理する BC 一様な論理回路族を $\mathbf{C} = \{C_n\}_{n \geq 0}$ とする．入力 1^n から $\overline{C_n}$ を生成する $\log n^c$ 領域限定の変換機を M_l とする．定理 4.1 より，M_l は多項式時間限定であるとしてもよい．次のように動作する DTM M を考える:

1. 長さ n の入力 $x = x_1 \cdots x_n$ が与えられたら，M_l を使って $\overline{C_n}$ を生成する．$|\overline{C_n}| = n^{O(1)}$ である．
2. $\overline{C_n}$ を使って，$x_1 \cdots x_n$ を入力として論理回路 C_n の計算を模倣する．この模倣は $O(|\overline{C_n}|^2)$ 時間でできる．これは n の多項式である．
3. $C_n(x_1, \ldots, x_n) = 1$ だったら x を受理し，$C_n(x_1, \ldots, x_n) = 0$ だったら拒否する．

M は多項式時間限定 DTM である．

(2) (1) のように $\overline{C_n}$ を生成してから模倣を始めると，$\overline{C_n}$ のために多項式サイズの領域量を使うことになる．そこで，$\overline{C_n}$ 全体を生成するのではなく，C_n の頂点を，出力ノードの側から入力ノードに向かって底優先探索するようにたどる (頂点は現在処理中のものを 1 つだけ覚えておき，その次に処理対象となる頂点は，現在覚えている頂点を使って入力 1^n から再計算して求める) ことによって C_n を模倣するようなアルゴリズム (DTM M) を考える．しかし，この模倣を補題 6.2 の証明のように直接的に行なうと，使う領域量が $O(\log^{k+1} n)$ になってしまう (1 つの頂点のコード $\overline{v_i} = [\overline{i}, type, \overline{j}, \overline{k}]$ を記録しておくために $O(\log n)$ 領域を使い，再帰の最大の深さが $O(\log^k n)$ であるから)．

そこで，再帰を使わない M を次のように構成する．上述したように，M は処理中の頂点を 1 つだけ (v_i とする) 覚えておく．また，出力ノードを始点として v_i へ至る道を 2 進数 $p \in \{0,1\}^*$ として記録しておく (0,1 は分岐の左右を表わす．NOT ゲートは分岐がないので 0 を対応させる)．また同時に，その道の上の各頂点 u に対し，u を根とする部分回路 (u が AND ゲートまたは OR ゲートなら 2 つあり，NOT ゲートなら 1 つしかない) の出力 (すでに計算が完了している場合には '0' または '1'，計算未了の場合にはそのことを表わすための '?') も記録しておく．v_i における出力を計算するためには，M は入力ノードに向かって道をさらに先にたどる必要がある．そのために，M は $O(\log n)$ 領域を使って $\overline{C_n}$ を再計算する．ただし，$\overline{C_n}$ の全体を生成することはせず，$\overline{v_i}$ を見て，必要な親ノード (のコード) $\overline{v_j}$ または $\overline{v_k}$ だけを生成する (余分な領域を使わないために，$\overline{v_i}$ は消去する)．v_i における出力 (0 または 1) が定まったら，v_i に対応する '?' をその値 ('0' または '1') に書き換え，さらに，p を使って v_i の子 (= v_i における出力が入力される頂点．w とする) を求める．w は入力 1^n と p から再計算し直す．それによって，w のタイプも w の親 (そのうち

の一つは v_i である) も分かる．w の親ノードにおける出力がすでにすべて求められていたら w における出力も定まるので対応する '?' を書き換え，w の子ノードを p を使って求める (w が出力ノードなら模倣は終了する)．そうでない場合には，出力の計算が未了の部分回路の計算に進む．

それぞれの入力ノードへの入力が何であるかを知るためには，入力ヘッドの位置を覚えておく必要があるが，これに使う領域量は $O(\log n)$ である．

以上の模倣で必要な領域量は $O(\log^k n + \log n)$ である．このうち，$O(\log n)$ は考慮中の頂点と入力ヘッド位置を覚えておくためのものであり，$O(\log^k n)$ は道 p とそれに付随するデータ ('0'/'1'/'?') の記録のために使われる領域量である． □

系 6.2 (1) $\mathbf{NC}^1 \subseteq \mathbf{DL}$. (2) $\mathbf{NC} \subsetneq \mathbf{PSPACE}$.

対数一様論理回路族の深さを対数の多項式に限定することによってクラス **NC** が定義されたが，この深さを制限しないとクラス **P** に一致する．

定理 6.7 対数一様論理回路族 (多項式サイズの BC 一様論理回路族) に受理される言語のクラスは **P** に等しい．

証明 包含関係 $\subseteq$ の証明は定理 6.6 (1) の証明と同じ．逆の包含関係は，定理 5.11 (5.3.1 節) の証明から得られる． □

問 6.8 厳密に言うと，定理 6.7 で述べている **P** の元は $\{0,1\}$ 上の言語だけである．しかし，簡潔な符号化のもとで，任意のアルファベット上の言語からなる **P** と $\{0,1\}$ 上の言語だけからなる **P** とを同一視することができる．このことを形式的に定義して証明せよ．また，クラス **NC** の場合にはどうか？

NC は，実際的なサイズの回路で非常に高速に並列計算できる問題のクラスとして重要である．$\mathbf{NC} \subseteq \mathbf{P}$ であるが，すべての実際的計算可能問題 (**P** の元) がこのように高速に並列計算できるか ($\mathbf{P} = \mathbf{NC}$ か) どうかは，重大な問題であるが現在のところ未解決である．それについて考えてみよう．

定理 6.8 (1) $L, L' \in \{0,1\}^*$ とする．$L \leq_{log} L'$ かつ $L' \in \mathbf{NC}$ ならば $L \in \mathbf{NC}$ である．もっと厳密に，次のことが成り立つ: $k \geq 2$ のとき，$L \leq_{log} L'$ かつ $L' \in \mathbf{NC}^k$ ならば $L \in \mathbf{NC}^k$ である．

(2) **NC** に属す **P** 完全問題が存在するならば $\mathbf{NC} = \mathbf{P}$ である．

証明 (1) $L \leq_{log} L'$ via M とする．次の条件を満たす $O(\log n)$ 領域限定の変換機 M' を構成することは容易である:

$$M(x) \text{ の第 } i \text{ ビットが } 1 \iff M' \text{ は入力 } (x,i) \text{ を受理する．}$$

ここに，i は $i \leq |M(x)|$ なる自然数 i の 2 進数表現である．$M(x)$ の第 i ビットは，M' の様相グラフ $G_{M',(x,i)}$ に対する到達可能性問題を解く (定理 4.3 の証明参照) ことによって $\mathbf{NC}^2$ で計算できる (例 6.5 (4) 参照)．よって，$M(x)$ の全ビットを並列に計算する $\mathbf{NC}^2$ 回路 A が存在し，その計算結果を，L' を受理する **NC** 回路 B に入力すれば $x \in L$ かどうかを計算できる．A と B を合わせた回路は多項式サイズかつ深さが $O(\log^2 n + \log^k n) = O(\log^k n)$ であるから，$L \in \mathbf{NC}^k$ である．

(2) 読者への演習問題とする． □

$\mathbf{NC} \subsetneq \mathbf{P}$ であろうと強く予想されている．というのは，その予想を裏付けるような事実がいくつもあるからである．例えば，一般の逐次アルゴリズムを模倣する並列アルゴリズムで現在知られているものはどれもさほどの高速化が達成されていない (もし $\mathbf{NC} = \mathbf{P}$ であるならば指数関数的に高速化できるはずなのに，逐次アルゴリズムで $t(n)$ 時間かかるものを模倣する並列アルゴリズムで現在知られているものはどれもたかだか $t(n)/\log t(n)$ 時間とか $\sqrt{t(n)}$ 時間といった程度の高速化にすぎない) こととか，いくつもの計算モデルのもとで **NC** や **P** に相当するクラスが等しくないことが証明されたり等しくないことが強く予想されたりする (例えば，TM のヘッド反転量について考えるとき，**NC** は対数多項式回のヘッド反転量のクラスに等しく **P** は多項式回のヘッド反転量のクラスに等しいことが知られているが*，ヘッド反転量の観点から見て，この 2 つのクラスは等しくないと強く予想される) こととか，もし $\mathbf{P} \subseteq \mathbf{NC}$ なら CVP が非常に少ない領域量 (対数多項式領域量) で計算できることになるが，CVP の計算に必要な領域量についてのさまざまな研究結果によると，そのことは否定的に思われること (あるゲートにおける値を計算するためには論理回路上でそのゲートより前にあるゲートでの値が計算されていなければならないので，それらの値を記憶しておくための領域が必要である．このような領域量が大きくなることを避けるためには，すべてのゲートの値を記憶しておくことをせず，あるゲートの値が必要になったときにうまく再計算ができるような方法があればよい．このことをモデル化したものにペブル法 (pebbling model) と呼ばれる一連の研究がある) †，などである．

では，**NC** に属する問題，すなわち，実際的な個数のプロセッサを使って非常に高速に並列計算ができる問題にはどのようなものがあるかを述べておこう．

---

*3.3 節および既出の N.J.Pippenger の論文 "On simultaneous resource bound" 参照．

†例えば，W.J.Paul, R.E.Tarjan and J.R.Celoni, Space bounds for a game on graphs, *Math. Sys. Theory* 10, pp.239-251, 1977.

H.Venkateswaran and M.Tompa, A new pebble game that characterizes parallel complexity classes, *SIAM J. Comput.* 18, pp.533-549, 1989.

例 6.5 (1) 例 6.3 (1) で見たように，n 変数パリティ関数を計算する，サイズ $O(n)$・深さ $O(\log n)$ の論理回路 C_n が存在する．入力 1^n から $\overline{C_n}$ を生成する $\log n$ 領域限定のアルゴリズムを示すのはむずかしいことではない．よって，パリティ関数は $\mathbf{NC}^1$ で計算可能である．また，問 6.7 より，ブール行列の積も $\mathbf{NC}^1$ で計算可能である．実は，多数決関数も $\mathbf{NC}^1$ で計算可能である．

(2) 例 6.4 より，$\mathrm{WW} \in \mathbf{NC}^1$ である．同様に，$\mathrm{WW}^R \in \mathbf{NC}^1$ である．証明はしないが，任意の文脈自由言語は $\mathbf{NC}^2$ に属することが知られている‡．

(3) 2 つの n ビット数の加算が $\mathbf{NC}^1$ で計算可能であることを示そう．例 6.3 で示した直列加算回路は $\mathbf{NC}^1$ ではないので，別の加算回路を考える．まず，桁上がり $c_1, \ldots, c_n$ (c_i は下位から i ビット目における上位のビットへの桁上がり) が並列に計算できることを示そう．

$$
\begin{aligned}
x_0 &= \mathbf{k}, \\
x_i &= \begin{cases} \mathbf{k} & a_i = b_i = 0 \text{ のとき} \\ \mathbf{g} & a_i = b_i = 1 \text{ のとき} \\ \mathbf{p} & a_i \neq b_i \text{ のとき} \qquad (i = 1, \ldots, n) \end{cases}
\end{aligned}
$$

と定義する ($\mathbf{k}$ は下位のビットからの桁上がりがあってもなくても桁上がりがそのビットで途切れる (kill) ことを，$\mathbf{g}$ は桁上がりが必ず生じる (generate) ことを，$\mathbf{p}$ は下位のビットからの桁上がりがあるかないかがそのまま上へ伝播する (propagate) ことを表わし，これらをそのビットの “桁上がり状態” と呼ぶ)．

$$y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_i \quad (i = 0, 1, \ldots, n)$$

と定義する．$\otimes$ は，連続する 2 ビット (x_i と x_{i+1}) 各々の桁上り状態に応じてその 2 ビット全体 $x_{i+1}x_i$ の桁上がり状態がどうなるかを表わす演算である:

	$\otimes$	x_{i+1}: $\mathbf{k}$	$\mathbf{p}$	$\mathbf{g}$
x_i	$\mathbf{k}$	$\mathbf{k}$	$\mathbf{k}$	$\mathbf{g}$
	$\mathbf{p}$	$\mathbf{k}$	$\mathbf{p}$	$\mathbf{g}$
	$\mathbf{g}$	$\mathbf{k}$	$\mathbf{g}$	$\mathbf{g}$

$x_i \otimes x_{i+1}$ の定義表
(可換でないことに注意)

$\otimes$ は結合律を満たし，$i \leq j$ に対して $[i,j] = x_i \otimes x_{i+1} \otimes \cdots \otimes x_j$ と定義すると，$(0 \leq i < k \leq j \leq n)$ なる任意の i, j, k に対して $[i,j] = [i,k-1] \otimes [k,j]$ が成り立つ．次のことを証明しよう:

$$
\begin{cases} y_i = \mathbf{k} \text{ ならば } c_i = 0. \\ y_i = \mathbf{g} \text{ ならば } c_i = 1. \\ y_i = \mathbf{p} \text{ となることはない}. \end{cases}
$$

‡W.L.Ruzzo, 1981: 対数一様性の脚注参照.

i に関する帰納法で証明する.

(基礎ステップ) $i = 0$ のときは $y_i = x_0 = \mathbf{k}$, $c_0 = 0$ であり, ok.

(帰納ステップ) y_i の値によって場合分けして考える.

$y_i = \mathbf{k}$ の場合, $y_i = y_{i-1} \otimes x_i$ であるから, $\otimes$ の定義表より,「$x_i = \mathbf{k}$ かつ y_{i-1} は任意」であるか「$x_i = \mathbf{p}$ かつ $y_{i-1} = \mathbf{k}$」である. $x_i = \mathbf{k}$ ならば, x_i の定義より $a_i = b_i = 0$ であり, したがって $c_i = \mathrm{MAJ}_3(a_i, b_i, c_{i-1}) = 0$ である. $x_i = \mathbf{p}$ かつ $y_{i-1} = \mathbf{k}$ ならば $a_i \neq b_i$ であり, 帰納法の仮定により $c_{i-1} = 0$ である. したがって, $c_i = \mathrm{MAJ}_3(a_i, b_i, c_{i-1}) = 0$ である.

$y_i = \mathbf{g}$ の場合,「$x_i = \mathbf{g}$ かつ y_{i-1} は任意」であるか「$x_i = \mathbf{p}$ かつ $y_{i-1} = \mathbf{g}$」である. $x_i = \mathbf{g}$ ならば $a_i = b_i = 1$ であるから $c_i = 1$ である. $x_i = \mathbf{p}$ かつ $y_{i-1} = \mathbf{g}$ ならば $a_i \neq b_i$ であり, 帰納法の仮定により $c_{i-1} = 1$ であるから $c_i = 1$ である.

$y_i = \mathbf{p}$ の場合, $\otimes$ の定義表より $y_{i-1} = \mathbf{p}$ でなければならないが, これは帰納法の仮定に反す.

以上で証明が終わった. この性質を使えば, y_i が高速に計算できるならば c_i も高速に計算できる. 実際, 性質 $[i, j] = [i, k-1] \otimes [k, j]$ を使って, $[i, j]$ と $y_{j-1} = [0, j-1]$ は, 図 6.11 のような論理回路 ◯ で計算できる. この論理回路に示されている $\otimes$ ゲートは実際は 4 入力 2 出力の論理回路で, $[\cdot, \cdot] = \mathbf{k}, \mathbf{p}, \mathbf{g}$ は, それぞれ 2 ビット 00, 01, 10 で表わされているものとする.

◯ を組み合わせて $y_0, y_1, \ldots, y_n$ を並列に計算する論理回路を図 6.12 に示した. $y_i = [0, i]$ のことを $x_0 \otimes x_1 \otimes \cdots \otimes x_n$ のプレフィックスといい, この回路は $x_0 \otimes x_1 \otimes \cdots \otimes x_n$ のすべてのプレフィックスを並列に計算するので, この回路のことをプレフィックス並列計算回路と呼ぶ. 図 6.13 に示した **kpg**

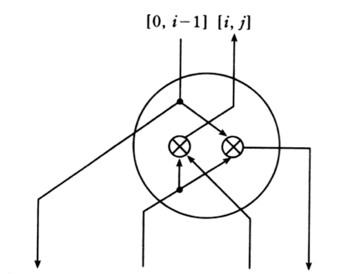


図 6.11 $[\cdot, \cdot]$ を計算する論理回路

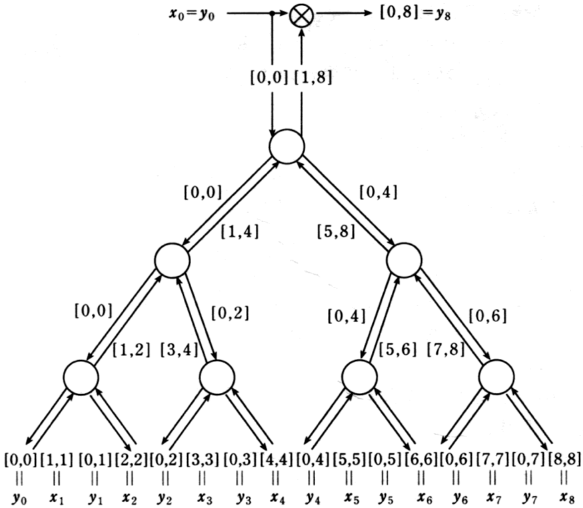


図 6.12 プレフィックス計算回路 ($n = 8$ の例)

回路とプレフィックス並列計算回路を使って，2 つの n ビット数 $a_n \cdots a_1$ と $b_n \cdots b_1$ の和 $s_{n+1}s_n \cdots s_1$ を求める論理回路 (**桁上がり先読み加算器** (carry look-ahead adder) という) を，図 6.14 に示した．この回路のサイズは $O(n)$，深さは $O(\log n)$ である．桁上げ先読み加算器の族が対数一様回路族であることの証明は省略する．以上で，n ビット数の加算は $\mathbf{NC}^1$ で計算可能であることが示された．

なお，n ビット数の減算や乗算は $\mathbf{NC}^1$ で計算可能，除算は $\mathbf{NC}^2$ で計算可能であることが知られている (減算については演習問題 8 参照) [§]．

(4) グラフの到達可能性問題 (GAP) を考えよう．$G = (V, E), V = \{1, 2, \ldots, n\}$ を有向グラフとする．$a_{ij} = 1 \Longleftrightarrow (i, j) \in E$ によって定義された $n \times n$ ブール行列 $A = (a_{ij})$ を G の**隣接行列** (adjacency matrix) という．$A^* = \sum_{i \geq 0} A^i$ を A の**反射推移閉包** (reflexive transitive closure) という．$A^* = \sum_{i=0}^{n-1} A^i$ である (問 6.9)．A^* の (i, j) 成分を a^*_{ij} とするとき，$a^*_{ij} = 1 \Longleftrightarrow$ 頂点 i から頂点 j

[§] A.Schönhage and V.Strassen, Schnelle Multiplication grosser Zahlen, *Computing* 7, pp.281-292, 1971.
J.H.Rief and S.R.Tate, Optimal size integer division circuits, *SIAM J. Comput.* 19, pp.912-924, 1990.

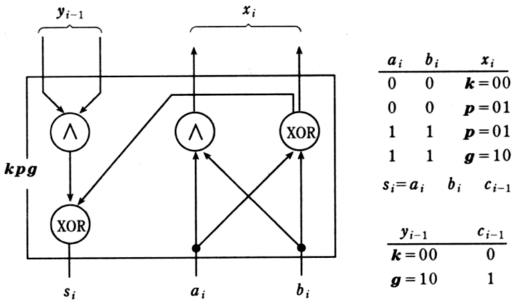


図 **6.13**　**kpg** 回路

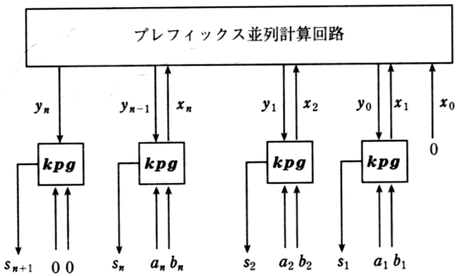


図 **6.14**　桁上がり先読み加算器

への道が存在する，が成り立つ (問 6.9)．よって，GAP はブール行列の反射推移閉包を求める問題 BOOLE–CLOSURE に還元される: GAP $\leq_{log}$ BOOLE–CLOSURE. 問 6.7 より，ブール行列の積は $\mathbf{NC}^1$ で求めることができる．一方，$A^n = A^{n/2}A^{n/2} = (A^{n/4}A^{n/4})(A^{n/4}A^{n/4}) = \cdots$ であるから，A^n は $\mathbf{NC}^1$ 回路を $\log n$ 段重ねた $\mathbf{NC}^2$ 回路 C_1 によって計算することができる．同様に，$A^* = \sum_{i=0}^{n-1} A^i$ も，$A^0, \ldots, A^{n-1}$ を入力とする $\mathbf{NC}^2$ 回路 C_2 で計算できる．よって，A^* は 2 つの $\mathbf{NC}^2$ 回路 C_1 と C_2 を重ねた $\mathbf{NC}^2$ 回路によって計算できる．　□

問 6.9 上例 (4) において，$A^k = (a_{ij}^{(k)})$ とするとき，$a_{ij}^{(k)} = 1 \iff i$ から j への長さ k の道が存在する，が成り立つことを k に関する帰納法で証明せよ．このことから，(4) で用いられた 2 つの事実が導かれる．

6.2.3 論理回路と交代性 TM

ここでは，ATM と論理回路の関係を考察しよう．以下で考える ATM の入力アルファベットは $\{0,1\}$ である (問 6.8 参照).

$\mathbf{C} = \{C_n\}_{n\geq 0}$ を論理回路族とする．C_n の任意の頂点 g のタイプ (すなわち，入力ノード/出力ノード/定数ノード/ゲートのどれであるかという情報と，ゲートの場合には素子の種類も情報として含む) を C_n のサイズに関して対数時間で計算できる DTM が存在するとき，$\mathbf{C}$ は E 一様であるという．もっと正確に定義しよう．$z(n) = \mathrm{size}(C_n)$ とする．次の言語を $\mathbf{C}$ の (頂点の間の) **結線記述** (extended connection language) という:

$$
\begin{aligned}
L_e(\mathbf{C}) = \{[n,s,p,t] \mid\ & n,s \in \{0,1\}^* \text{ で } s \text{ は } C_n \text{ の頂点番号,}\\
& p \in \{\text{`L'}, \text{`R'}\}^*,\ |p| \leq \log z(n), \text{ かつ}\\
& p = \varepsilon \text{ なら } t = \mathit{type}(p) \in \{0, 1, \text{``}\mathit{in}\text{''}, \text{``}\mathit{out}\text{''}, \neg, \wedge, \vee\},\\
& p \neq \varepsilon \text{ なら } t \text{ は } C_n \text{ の頂点番号で } t = p(s) \in \{0,1\}^*\}.
\end{aligned}
$$

C_n 内の道を $\{\text{`L'}, \text{`R'}\}^*$ の元で表わす (‘L’/‘R’ は左分岐/右分岐を表わす．ただし，分岐がないときは ‘L’ で表わす)．また，頂点 s から出発し p に従って道を先祖に向かってたどったときに，最後に到達する頂点を $p(s)$ で表わす．とくに，$p = \varepsilon$ なら $p(s) = s$ である．$L_e(\mathbf{C})$ の元 $[n,s,p,t]$ は，C_n において，番号 s の頂点から出発し p に従って道を先祖 (＝入力側) に向かってたどったとき，最後に到達する頂点が t であることを表わしている．$p = \varepsilon$ の場合，t はその頂点のタイプを表わし，$p \neq \varepsilon$ の場合，t はその頂点の番号を表わす．とくに，C_n の出力ノードは $[n, 0, \varepsilon, \text{``}\mathit{out}\text{''}]$，入力ノード $x_i$ は $[n, \bar{i}, \varepsilon, \text{``}\mathit{in}\text{''}]$ なる $L_e(\mathbf{C})$ の元で表わされているものとする (C_n の頂点は 0 からの通し番号が付けられるわけではないが，頂点番号の 2 進数表現の長さは $O(\log z(n))$ でなければならないという制約はある)．$|p| \leq \log z(n)$ という条件は，$L_e(\mathbf{C})$ によって記述されているのは深さがたかだか $\log z(n)$ 離れた頂点間の結線具合だけである (むしろ，$L_e(\mathbf{C})$ は $\log z(n)$ も離れた頂点間の結線具合までも記述しているというべきかもしれない) ことを示している．$L_e(\mathbf{C})$ は，対数一様性や BC 一様性を定義する際に回路を記述するために用いた $\{\overline{C_n} \mid n \geq 0\}$ (この言語は，深さが 1 だけ離れた頂点間の結線しか記述していない) の拡張である．

$L_e(\mathbf{C})$ を受理する $O(\log z(n))$ 時間限定の DTM が存在するならば，$\mathbf{C}$ は **E 一様** (E-uniform) であるという．また，$d(n) = \mathrm{depth}(C_n)$ とするとき，$L_e(\mathbf{C})$ を受理する $O(d(n))$ 時間限定かつ $O(\log z(n))$ 領域限定の ATM が存在するならば，$\mathbf{C}$ は **E*一様** (E*-uiniform) であるという．クラス

$$\mathbf{U}_E\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(z(n), d(n)),\quad \mathbf{U}_{E^*}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(z(n), d(n))$$

を $\mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(z(n), d(n))$ と同様に定義する．

補題 6.7 (1) $\mathbf{C}$ が E 一様なら $\mathbf{C}$ は BC 一様である.

(2) $\mathbf{C}$ が E 一様なら $\mathbf{C}$ は E*一様である.

(3) $\mathbf{C}$ が BC 一様かつ $d(n) \geq \log^2 z(n)$ なら $\mathbf{C}$ は E*一様である.

証明 (1) $\mathbf{C}$ が E 一様であるとすると，$L_e(\mathbf{C})$ を受理する $\log z(n)$ 時間限定の DTM が存在する．長さ n の入力に対し，C_n のコード $\overline{C_n}$ を出力する変換機 $\overline{M}$ を構成する．$\overline{M}$ は C_n 内の頂点 s すべてについて，そのコード $\overline{s}$ を次のように求めて出力する (以下のプログラムにおいて，s, s_L, s_R は 2 進数 ($\in \{0,1\}^*$) である):

```
max_gate_No ← 0; s ← 0;
while s ≤ max_gate_No do
    if [n, s, ε, t] ∈ L_e(C) なる t ∈ {0, 1, "in", ¬, ∧, ∨} が存在する then
        if t = 0 or t = 1 or t = "in" then
            s̄ = [s, t, 0, 0] を出力する
        else if t = "out" or t = ¬ then
            [n, s, 'L', s_L] ∈ L_e(C) なる s_L を見つける;
            /* s_L は s の唯一の親頂点 */
            s̄ = [s, t, s_L, 0] を出力する
        else if t = ∧ or y = ∨ then
            [n, s, 'L', s_L] ∈ L_e(C) なる s_L を見つける;
            [n, s, 'R', s_R] ∈ L_e(C) なる s_R を見つける;
            /* s_L, s_R は s の左右の親頂点 */
            s̄ = [s, t, s_L, s_R] を出力する
        end end end
    end;
    s ← s + 1;
    max_gate_No ← max{s_L, s_R, max_gate_No}
end
```

s の親 s' を見つける部分は次のように行なう (左の親のときは $p =$ 'L'，右の親のときは $p =$ 'R'):

```
/* [n, s, p, s'] ∈ L_e(C) なる s' を見つける */
s' ← 0;
while [n, s, p, s'] ∉ L_e(C) do s' ← s' + 1 end
```

仮定より，$L_e(\mathbf{C})$ の元であるかどうかの判定は $O(\log z(n))$ 領域でできる．また，$\overline{M}$ は出力した頂点の個数をカウンタに記録しておく．このカウンタに使う領域は $O(\log z(n))$ あれば十分である (while 文はたかだか $z(n)$ 回しか繰り返さないことに注意する)．$\overline{M}$ は $\overline{C_n}$ を $O(\log z(n))$ 領域しか使わずに出力することができるので，C は BC 一様である．

(2) $d(n) = \Omega(\log z(n))$ であることを考慮すると，定義より明らか (DTM は ATM の特別の場合であることに注意する)．

(3) $\overline{C_n}$ を生成する $O(\log z(n))$ 領域限定DTMを $\overline{M}$ とする．$L_e(\mathbf{C})$ を受理する $O(\log z(n))$ 領域限定 DTM M を次のように構成する．入力 x が与えられたとき M はまず，x が正しい形

$$[n, s, p, t]$$

(ここに，$n, s \in \{0,1\}^*, p \in \{\text{'L'}, \text{'R'}\}^*, t \in \{0, 1, \text{"}in\text{"}, \text{"}out\text{"}, \neg, \wedge, \vee\} \cup \{0,1\}^*$) をしているかどうかをチェックし，正しい形の入力 $[n, s, p, t]$ に対してだけ，次の手続き L_e を呼び出す:

```
procedure L_e(n, s, p, t)
begin
    while p ≠ ε do
        p = qr, q ∈ {'L', 'R'} とする;
        if r = ε then
            if q(s) = t then 入力を受理する else 入力を拒否する end
        else
            L_e(n, q(s), r, t)
        end
    end;
    /* ここでは p = ε である */
    if t が s のタイプと一致する then 入力を受理 else 入力を拒否 end
end
```

$q(s) = t$ を求めるときと，t が s のタイプと一致するかどうか調べるとき，M は $[s, type, s_L, s_R]$ が出力されるまで $\overline{M}$ を模倣する ($type$ は s のタイプ，s_L/s_R は s の左/右の親頂点)．M は自分自身を再帰的に呼び出す (たかだか $|p| \leq \log z(n)$ 回) つど同じ領域を再利用して計算を進めることにより，$O(\log z(n))$ 領域限定で動作するようにできる．演習問題 4 (c) より，M を模倣する $O(\log z(n))$ 領域限定かつ $O(\log^2 z(n))$ 時間限定の ATM が存在する．仮定 $d(n) \geq \log^2 z(n)$ を考慮すると，(3) が導かれる． □

論理回路と ATM の関係を述べる手始めとして，まずはじめに，ATM を論理回路で模倣する．

補題 6.8 $s(n) = \Omega(\log n)$ とし，$t(n)$ と $s(n)$ の 2 進数表現は $s(n)$ 時間で構成可能とする．このとき，$\mathbf{ATISP}(t(n), s(n)) \subseteq \mathbf{U}_E\text{-}\mathbf{SIZE} \cdot \mathbf{DEPTH}(2^{O(s(n))}, t(n))$ である．

証明 M を $t(n)$ 時間限定かつ $s(n)$ 領域限定のオフライン ATM とする．命題6.1 より，M の作業テープは 1 本であると仮定してもよい．また，M の 1 ステップは次のいずれかであると仮定しても一般性を失わない．

- 入力を受理または拒否して停止する．

- 全称 ID または存在 ID から次の ID へ遷移する．次の ID はちょうど 2 個である．
- 入力テープ上の 1 こまを読み，別の ID に一意的に遷移する．

$L(M)$ を受理する E 一様な論理回路族 $\{C_n\}_{n \geq 0}$ を構成する．長さ n の入力 $x_1 \cdots x_n$ $(x_i \in \{0,1\})$ に対する M の計算木を模倣するように，論理回路 C_n を構成する．

- C_n の入力ノードは $x_1, \ldots, x_n$.
- C_n の入出力・定数ノード以外の全頂点に (T, I) なるラベルを付ける (注：こうしたラベルは C_n の頂点を一意的に表わしており，$L_e(\mathbf{C})$ の元 $[n, \cdot, \cdot, \cdot]$ においては頂点番号としてコード化されるものとする)．I は M の ID の 1 つであり，T は M が I に到達した時刻 $(0 \leq T \leq t(n))$ を表わしているか，または M が ID I において入力テープから記号 0/1 を読もうとしていること (このことを，"read 0" または "read 1" で表わす) を表わしている．ただし，ここでいう ID とは (q, i, α) のことである．q は状態，i は入力ヘッド位置の 2 進数表現，α は作業テープの内容である．
- 以下の構成において，ゲート g の出力が直接，ゲート g' の入力になるとき，g を g' の**親** (predecessor) という．

  ラベル $(0, I_0)$ を付けた頂点を C_n の出力ノード (1 つしかない) の親とする．ここに，I_0 は M の初期 ID である．
- 以下で，g は (T, I_i) がラベル付けされた，C_n の頂点 (入出力ノード，定数ノード以外) とする．

  1. M が時刻 T に ID I_i から $\forall$ に I_j と I_k に遷移するとき，g にはゲート $\wedge$ を割り当て，g の左の親と右の親にそれぞれ $(T+1, I_j)$, $(T+1, I_k)$ をラベル付けする (図 6.15 (a))．とくに，$T+1 > t(n)$ であるか，または I_j (あるいは I_k) の領域量が $s(n)$ を越えている場合には，それらは定数ノード 0 とする．
  2. M が時刻 T に ID I_i から $\exists$ に I_j と I_k に遷移するとき，g にゲート $\vee$ を割り当てる以外は 1 と同じ (図 6.15 (b))．
  3. M が時刻 T に入力文字 $x_s \in \{0,1\}$ を読んで ID I_i から ID I_j へ ($x_s = 0$ のとき) または I_i から I_k へ ($x_s = 1$ のとき) 遷移するとき，g にはゲート $\vee$ を割り当て，新たに頂点 g_L と g_R を作り，それぞれ g の左と右の親とする．g_L, g_R にはそれぞれ $\wedge$ を割り当てる．さらに，g_L の左の親となる頂点と g_R の右の親となる頂点を作り，

それらにそれぞれ $(T+1, I_j)$, $(T+1, I_k)$ をラベル付けする．また，入力ノード x_s を g_R の左の親とし，g_L の右の親として新しく頂点 g' を作り $\neg$ を割り当てる．x_s を g' の親とする．g_L, g_R, g' にはそれぞれラベル (“read 0”, I_i), (“read 1”, I_i), (“read 1”, I_i) を付ける (図 6.15 (c)).

4. I_i が受理 ID のとき，g を定数ノード 1 とする．
5. I_i が拒否 ID のとき，g を定数ノード 0 とする．

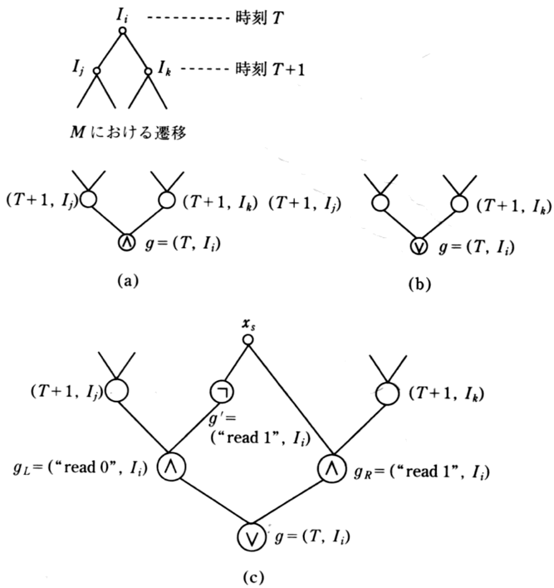


図 6.15 M の遷移とそれに対応する C_n の部分

C_n の頂点のうち，入出力ノードは合わせて $n+1$ 個である．長さ n の入力に対する M の ID の個数はたかだか $2^{O(s(n))}$ であるから，時刻と ID との対がラベルとして付いているゲートはたかだか $t(n)2^{O(s(n))}$ 個である．また，定数ノードも，“read 0” または “read 1” と ID との対がラベル付けされている AND ゲートまたは NOT ゲートも，それぞれたかだか $t(n)2^{O(s(n))}$ 個しかない．よって，$\mathrm{size}(C_n) \le n+1+4t(n)2^{O(s(n))} = 2^{O(s(n))}$ である．最後の等号が成り立つために，条件 $s(n) \ge \log n$ と，$t(n)$ の 2 進数表現が $s(n)$ 時間で構成可能

こと ($\log_2 t(n) = O(s(n))$) が必要である．$\mathrm{depth}(C_n) \le 2t(n)+2 = O(t(n))$ であることは明らかであろう．また，C_n の構成の仕方より，$L(\mathbf{C}) = L(M)$ であることも明らかである (読者への問とする)．

最後に，$\mathbf{C} = \{C_n\}_{n\ge 0}$ がE一様であることを示すために，$L_e(\mathbf{C})$ を受理する，$O(s(n))$ 領域限定のDTMが存在することを示せばよい．上記のように構成した C_n の入出力・定数ノード以外の各頂点には (T, I) なる形のラベルが付いているが，$[n,s,p,t] \in L_e(\mathbf{C})$ なる $s, t \in \{0,1\}^*$ は，このラベルを符号化したものである．そのラベルを見れば，頂点のタイプが分かることに注意する．次のDTM M_e を考える:

```
procedure Mₑ(x)
begin
    if x が [n,s,p,t] なる形をしていない then 入力 x を拒否して停止する end;
    /* 以下，x = [n,s,p,t] とする */
    if x が正しい入出力ノードの形をしている
        then x を受理して停止 else 拒否して停止
    end;
    s を復号してラベル (T,I) を取り出し，I から s のタイプを取り出す;
    if p = ε then
        if 取り出したタイプが t と一致する then x を受理 else 拒否 end
    else
        if T が時刻を表わしている then
            I を出発点として |p| ステップだけ M の動作を模倣する
            /* (I ⊢_M^{|p|} J だとする);
                M の各ステップは ∀ に，あるいは ∃ に2分岐するが，
                Mₑ は p に従ってどちらか一方だけを選ぶ */
            if ラベル (T+|p|, J) が t と一致する then x を受理 else 拒否 end
        else
            I が "read 0" や "read 1" である場合も同様
        end
    end
end
```

M は $O(s(n))$ 領域限定だから，M_e は $I \vdash_M^{|p|} J$ の模倣を $O(s(n))$ 領域で行なうことができる．また，仮定より，$t(n), s(n)$ は $s(n)$ 時間で構成可能であるから $|p| \le \log_2 t(n) = O(s(n))$ であり，したがって，M_e は模倣すべきステップ数 $|p|$ を越えたかどうかを $O(s(n))$ 領域でチェックできる．よって，M_e は $O(s(n))$ 領域限定である． □

今度は逆に，論理回路をATMで模倣する．

補題 6.9 $d(n) = \Omega(\log n)$ かつ $z(n) = n^{\Omega(1)}$ とするとき，U_{E^*}-**SIZE** · **DEPTH** $(z(n), d(n)) \subseteq$ **ATISP**$(d(n), \log z(n))$ である．

証明 $\mathbf{C} = \{C_n\}_{n\geq 0}$ を E*一様回路族とする．仮定より，$L_e(\mathbf{C})$ を受理する $O(d(n))$ 時間限定かつ $O(\log z(n))$ 領域限定の ATM M_e が存在する．$x = x_1 \cdots x_n$ $(x_i \in \{0,1\})$ を入力すると $C_n(x)$ を計算し，$C_n(x) = 1/0$ に従って x を受理/拒否する (すなわち，$\mathbf{C}$ を受理する) $O(d(n))$ 時間限定かつ $O(\log z(n))$ 領域限定の ATM を構成したい．次の ATM M_{CV} を考える．

```
procedure M_CV(x, g, p)
begin
    n ← |x|;
    ∃ に次の 1, 2 を実行する;
    1. /* |p| < log z(n) と推測した場合 */
        ∃ に y ∈ {0, 1, "in", ∨, ∧, ¬} を選ぶ;
        if [n, g, ε, y] ∈ L(M_e) then
            if y = 1 then x を受理して停止する end;
            if y = 0 then x を拒否して停止する end;
            if y = "in" and x_g = 1 then x を受理して停止 end;
            if y = "in" and x_g = 0 then x を拒否して停止 end;
            if y = ¬ then M̅_CV(x, g, p'L') を実行する else
            if y = ∧ then
                ∀ に M_CV(x, g, p'L') と M_CV(x, g, p'R') を実行する else
            if y = ∨ then
                ∃ に M_CV(x, g, p'L') と M_CV(x, g, p'R') を実行する
            end end end
        end;
    2. /* |p| = log z(n) と推測した場合 */
        ∃ に g' ∈ {0, 1}* を推測する;
        if [n, g, p, g'] ∈ L(M_e) then M_CV(x, g', ε) end
end
```

$M_{CV}(x,g,p)$ が受理で停止 $\Longleftrightarrow$ 入力 x に対し, $C_{|x|}$ の頂点 $p(g)$ における値が 1

が成り立つ．とくに，ATM $M = M_{CV}(\cdot, 0, \varepsilon)$ は $\mathbf{C}$ を受理する．

$\overline{M}_{CV}$ は M_{CV} と双対的な ATM で，M_{CV} を次のように修正することによって得られるものである:「受理」と「拒否」,「∀ に」と「∃ に」, M_{CV} と $\overline{M}_{CV}$ をそれぞれ入れ替える．このとき，

$\overline{M}_{CV}(x,g,p)$ が受理で停止 $\Longleftrightarrow$ 入力 x に対し, $C_{|x|}$ の頂点 $p(g)$ における値が 0

が成り立つ．

M において M_{CV} と $\overline{M}_{CV}$ が呼ばれる深さはたかだか $O(d(n))$ であり，仮定から，M_e の模倣は $O(d(n))$ 時間限定の ATM で行なうことができるから，$[n,g,\varepsilon,y] \in L(M_e)$ の計算と **if** $y = \cdots$ それぞれの計算とを ∀ に行なうように M_{CV} を修正すれば，M を $O(d(n))$ 時間限定とすることができる．また，M_e は $O(\log z(n))$ 領域限定の ATM であるから，M_{CV} が再帰呼び出し先でも同じ

$O(\log z(n))$ 領域をいつも再利用するようにすれば (M_{CV} は ATM であるから，これが可能である)，M の領域量を $O(\log z(n))$ とすることができる．　□

補題 6.8 ∼ 6.9 より，次の定理が得られる¶．

定理 6.9 任意の $k \geq 2$ に対して $\mathbf{NC}^k = \mathbf{ATISP}(\log^k n, \log n)$ である．

証明 次のように，一様性が異なる 3 つのクラスと $\mathbf{ATISP}(\log^k n, \log n)$ が一致する:

$$\begin{array}{lll}
\mathbf{NC}^k & = & \mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)}, \log^k n) \quad (\mathbf{NC}^k\text{の定義}) \\
& \subseteq & \mathbf{U}_{E^*}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)}, \log^k n) \quad (\text{補題 6.7 (3)}) \\
& \subseteq & \mathbf{ATISP}(\log^k n, \log n) \quad (\text{補題 6.9}) \\
& \subseteq & \mathbf{U}_{E}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(2^{O(\log n)}, \log^k n) \quad (\text{補題 6.8}) \\
& = & \mathbf{U}_{E}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)}, \log^k n) \\
& \subseteq & \mathbf{U}_{BC}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(n^{O(1)}, \log^k n) \quad (\text{補題 6.7 (1)}) \\
& = & \mathbf{NC}^k.
\end{array}$$
□

系 6.3 $\mathbf{NC} = \mathbf{ATISP}(n^{O(1)}, (\log n)^{O(1)})$.

NC とヘッド反転量 (3.3 節参照) との関係については $\mathbf{NC} = \mathbf{DTIRE}\,(n^{O(1)}, (\log n)^{O(1)})$ であることが知られている．

NC は深さが "浅い" 論理回路であるのに対して，幅が "狭い" 論理回路を考えることができる．C を論理回路とする．C の定数ノードと入力ノードの集合を G_0 とする．$i > 0$ に対し，

$$G_i = \{C \text{ の頂点 } v \mid v \notin \bigcup_{j<i} G_j \text{ かつ } v \text{ の親頂点は} \bigcup_{j<i} G_j \text{ の元}\}$$

と定義する．$\max_i |G_i|$ を C の**幅** (width) という．

$$\begin{array}{lll}
\mathbf{SC}^k & = & \{L(C) \mid C \text{ は幅が } O(\log^k n) \text{ の多項式サイズ論理回路}\}, \\
\mathbf{SC} & = & \bigcup_k \mathbf{SC}^k
\end{array}$$

と定義する‖．定義より，$\mathbf{DL} = \mathbf{SC}^1 \subseteq \mathbf{SC}^2 \subseteq \cdots \subseteq \mathbf{SC} \subseteq \mathbf{P}$ である．$\mathbf{NL} \subseteq \mathbf{SC}$ かどうかは分かっていない．

命題 6.2 $k \geq 1$ に対して，

$$\mathbf{SC}^k \subseteq \mathbf{DTISP}(n^{O(1)}, (\log n)^{O(1)}), \quad \mathbf{DTISP}(n^{O(1)}, \log^k n) \subseteq \mathbf{SC}^k.$$

したがって，$\mathbf{SC} = \mathbf{DTISP}(n^{O(1)}, (\log n)^{O(1)})$ である．

¶W.L.Ruzzo, On uniform circuit complexity, *J. Comput. Sys. Sci.* 22, pp.365-383, 1981.

‖SC は "Steve's class" の頭文字である．このクラスの重要性に初めて気付いた Stephen Cook を顕彰して N.Pippenger が命名した．NC についての脚注参照．S.Cook と N.Pippenger の論文は同ページの脚注に示した．命題 6.2 については，Pippenger の論文を参照のこと．

6.3 並列ランダムアクセス機械

並列計算の3つめのモデルとして，この節では，もっとも自然で広く用いられている一つである**並列ランダムアクセス機械** (parallel random access machine, **PRAM** と略記する) について考察する**. これは，同じタイプの複数のプロセッサが，メモリを共有しながら同期して動作するものである (図 6.16). 一つ一つのプロセッサを **RAM** といい，**共有メモリ** (common global memory) 以外に各 RAM は固有の**局所メモリ** (local memory) を持っている．RAM という名前は，メモリのどの部分から/へのデータの読み/書きも同じ一定時間でできるというところに由来する (これに対し，TM のテープは**逐次アクセスメモリ** (sequential access memory) である).

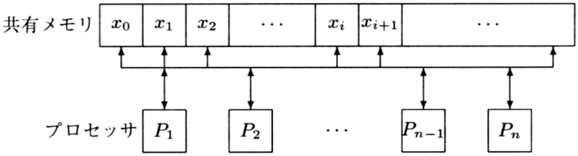


図 6.16 PRAM

共有メモリは可算無限個の，**レジスタ** (register) と呼ばれるセル $x_0, x_1, \ldots,$ $x_i, \ldots$ からなる．各レジスタは1つの (任意に大きい) 非負整数を記憶でき，それらは0と1の記号列として2進数で表わされているものとする．各プロセッサは図 6.17 に示したように，演算装置と，可算無限個のレジスタ $y_0, y_1, \ldots, y_i, \ldots$ からなる局所メモリとを持っており，プログラム領域にプログラムを保持している．局所メモリの各レジスタも任意の非負整数を記憶できる．x_i や y_i の添字 i をレジスタの**アドレス** (address) という．

すべてのプロセッサは同じクロックで制御されており，各プロセッサは1単位時間毎の同一時刻に同期して，それぞれのプログラム内の1つの命令を実行する．各プロセッサが保持しているプログラムは同一である．このようなタイプの PRAM においては，どのプロセッサも同じ時刻に1つの同じ命令を異なるデータに対して実行する (どのプロセッサも同じプログラムを実行するのに，なぜプロセッサごとに異なるデータに対して実行することができるのかについ

**S.Fortune and J.Wyllie, Parallelism in random access machines, *Proc. 10th ACM Symp. on Theory of Computing*, pp.114-118, 1978.

L.M.Goldschlager, A unified approach to models of synchronous parallel machines, *Proc. 10th ACM Symp. on Theory of Computing*, pp.89-94, 1978.

W.J.Savitch and M.Stimson, Time bounded random access machines with parallel processing, *J. Assoc. Comput. Mach.* 26, pp.103-118, 1979.

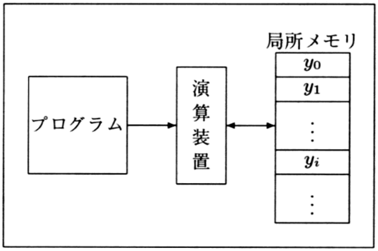


図 **6.17**　RAM プロセッサ

ては後述する) という意味で **SIMD** (single instruction multiple data) **型**であるという．これに対し，同じ時刻にプロセッサごとに異なる命令が実行できるようなモデルを (**MIMD** (multiple instruction multiple data) **型**という．本書では SIMD 型 PRAM モデルを採用するが，両モデルに本質的な差はない．

6.3.1　PRAM プログラム

プログラムは，以下に述べる命令 (**基本命令**) を使って書く．以下の説明において，i, j, k は任意の非負整数とする．また，$y[y_i]$ は y_{y_i} を表わす．

1. **プロセッサ番号識別命令**

(1)　$y_i \leftarrow$ プロセッサ番号

この命令の意味については後述する．

2. **代入および演算命令**

(2)	$y_i \leftarrow c$	(3)	$y_i \leftarrow y_j$
(4)	$y_i \leftarrow y_j + y_k$	(5)	$y_i \leftarrow y_j - y_k$
(6)	$y_i \leftarrow y_j * y_k$	(7)	$y_i \leftarrow y_j / y_k$
(8)	$y_i \leftarrow\uparrow y_j$	(9)	$y_i \leftarrow\downarrow y_j$
(10)	$y_i \leftarrow y_j \vee y_k$	(11)	$y_i \leftarrow y_j \wedge y_k$

(2) において，c は任意の自然数である．(5) において，$y_j \leq y_k$ のときは $y_j - y_k = 0$ であると定義する (負の整数は扱わない)．(7) において，y_j/y_k は y_j を y_k で割った商，すなわち $\lfloor y_j/y_k \rfloor$ を表わす．(8), (9) において，$\uparrow, \downarrow$ はレジスタの内容をそれぞれ左/右に 1 ビットシフトする演算である．したがって，$\uparrow y_j$ は $2y_j$ に等しく，$\downarrow y_j$ は $\lfloor y_j/2 \rfloor$ に等しい．(10), (11) において，$\vee$ と $\wedge$ はそれぞれビットごとの論理和，論理積をとる演算である．ただし，y_j, y_k の先頭に 0 を必要なだけ付け加えて同じ長さにして演算が行なわれる．

2. メモリの読み書き命令

(12)　$y_i \leftarrow y[y_j]$　　(13)　$y_i \leftarrow x[y_j]$

(14)　$y[y_j] \leftarrow y_i$　　(15)　$x[y_j] \leftarrow y_i$

(12), (13) は，y_j の値をアドレスとするレジスタの内容をレジスタ y_i に読み込む命令であり，(14), (15) は，y_j の値をアドレスとするレジスタにレジスタ y_i の内容を書き込む命令である．このように，対象になるレジスタのアドレスが他のレジスタの内容によって定まるアドレス決定方式を**間接番地方式** (indirect addressing) という．y_j として，レジスタ以外に定数 (任意の自然数) も許すことにする．

3. ジャンプ命令

(16)　**goto** ℓ

(17)　**goto** ℓ **if** $y_i = 0$　　(18)　**goto** ℓ **if** $y_i \neq 0$

(16) が実行されると，無条件に ℓ 番目の命令にジャンプする．一方，(17), (18) では，**if** に記された条件件が満たされたときだけ ℓ 番目の命令にジャンプする．

4. 停止命令

(19)　**halt**

上述の命令は必要最小限のものではなく，他の命令の組み合わせで記述できるような命令も多く含まれている．**プログラム**はこれらの命令を 1 行に 1 つずつ書いたもので，1 行目から順に実行される．ただし，ジャンプ命令を実行すると実行順序が変更される．停止命令に出会うと，実行を終了する．

各プロセッサ (RAM) は同じプログラムを実行するが，各 RAM はプロセッサ番号識別命令によって自分のプロセッサ番号を知ることができ，それによってプログラムの実行を制御できるので，同一のプログラムでもプロセッサごとに異なる仕事をすることができる．例えば，

1. $y_0 \leftarrow$ プロセッサ番号
2. $y_1 \leftarrow\downarrow y_0$　/∗ $y_1 \leftarrow y_0$を 2 で割った商 ∗/
3. $y_1 \leftarrow\uparrow y_1$
4. $y_1 \leftarrow y_0 - y_1$　/∗ $y_1 =$「y_0 を 2 で割った余り」となる ∗/
5. **goto** 9 **if** $y_1 = 0$　/∗ $y_1 = 0$ (y_0が偶数) なら何もしない ∗/
6. $y_2 \leftarrow x[y_0]$
7. $y_2 \leftarrow\uparrow y_2$　/∗ y_2を 2 倍する ∗/
8. $x[y_0] \leftarrow y_2$　/∗ $x[y_0] =$「もとの $x[y_0]$ の値の 2 倍」となる ∗/
9. **halt**

の場合，奇数番号のプロセッサだけが，共有メモリ上のレジスタのうち自分のプロセッサ番号と同じアドレスのレジスタの内容を 2 倍する．

人間にとって RAM プログラムが読み易くなるように，以下に述べるような略記法を用いる．まず，共有メモリや局所メモリの“適当な”レジスタを記号名で (例えば，$x_{10}, y_{100}, x_{10}[y_{100}]$ をそれぞれ $a, b, a[b]$ で) 表わし，**変数** (variable) と呼ぶ．変数は，そのレジスタに保持されている値を表わす．共有メモリを使う変数を**大域的変数**，局所メモリを使う変数を**局所変数**という．

i, j, m を変数 (m は任意の自然数でもよい) とし，$k, \ell \geq 1$ を任意の自然数とするとき，

$$\textbf{for } \mathrm{i} \leftarrow 1 \textbf{ to } \mathrm{m} \textbf{ do } S_1; \ldots ; S_k \textbf{ end}$$

は

$\ell.$	$\mathrm{i} \leftarrow 1$	i は変数，例えば $\mathrm{i} = y_0$
$\ell+1.$	**goto** $\ell+k+7$ **if** $\mathrm{i} > \mathrm{m}$	これは以下の 3 行の省略形である： $\ell+1.\ \mathrm{j} \leftarrow \mathrm{m}$ $\ell+2.\ \mathrm{j} \leftarrow \mathrm{i} - \mathrm{j}$ $\ell+3.$ **goto** $\ell+k+7$ **if** $\mathrm{j} \neq 0$
$\ell+4.$	S_1	$S_1, \ldots, S_k$ は基本命令であるとする
	$\cdots$	
$\ell+k+3.$	S_k	
$\ell+k+4.$	$\mathrm{i} \leftarrow \mathrm{i} + 1$	これは以下の 2 行の省略形である： $\ell+k+4.\ \mathrm{j} \leftarrow 1$ $\ell+k+5.\ \mathrm{i} \leftarrow \mathrm{i} + \mathrm{j}$
$\ell+k+6.$	**goto** $\ell+1$	
$\ell+k+7.$	$\cdots$	

を略記したものである．同様に，

$$\textbf{if } \text{条件式} \textbf{ then } S_1; \ldots ; S_k \textbf{ end}$$

などの省略形が定義されるが，詳細は省略する．

問 6.10 x, y を変数，$S_1, \ldots, S_k, S'_1, \ldots, S'_\ell$ をそれぞれ基本命令とするとき，

$$\textbf{if } x = y \textbf{ then } S_1; \ldots ; S_k \textbf{ else } S'_1; \ldots ; S'_\ell \textbf{ end}$$

を本来の PRAM プログラムとして書け．

例 6.6 $s = \sum_{i=1}^{n} a_i$ を求める並列アルゴリズム (PRAM プログラム) を考える．入力として，共有メモリ $x_1, \ldots, x_n$ に自然数 $a_1, \ldots, a_n$ が格納されているものとする．$x_1, \ldots, x_n$ を $\mathrm{x}[1], \ldots, \mathrm{x}[n]$ で，$x_{n+1}, \ldots, x_{2n}$ を $\mathrm{z}[1], \ldots, \mathrm{z}[n]$ で表わす．$\mathrm{x}[1], \ldots, \mathrm{x}[n]$ と $\mathrm{z}[1], \ldots, \mathrm{z}[n]$ はそれぞれメモリ内の連続した n 個のレジスタを占めている．このような x や z を**配列** (array) と呼ぶ．x, z のように，共有メモリを使う配列を大域的配列という．$\mathrm{z}[i]$ は配列 z の第 i 要素であり，**添字付き変数**ともいう．[ ] は間接番地表示と矛盾していない．配列の添字の範囲は明示せず，文脈から判断するものとする．出力は大域的変数 s に格納する．以下のアルゴリズムでは n 個のプロセッサ $P_1, \ldots, P_n$ を使う．

```
i ← プロセッサ番号;
z[i] ← x[i];   /* z[i] は大域的変数 */
for h ← 1 to ⌈log₂n⌉ do
    if i ≤ n/2^h then z[i] ← z[2i − 1] + z[2i] end
end;
s ← z[1]
```

i がプロセッサ番号を保持する変数のとき (すなわち,

$$i \leftarrow \text{プロセッサ番号}$$

という命令が実行された後), i を含む命令はプロセッサ i だけが実行する. したがって, 上のプログラムの $z[i] \leftarrow x[i]$ は P_i が実行し, $z[i] \leftarrow z[2i-1]+z[2i]$ は $i \le n/2^h$ を満たすプロセッサ i だけが実行する.

$$\textbf{for } h \leftarrow 1 \textbf{ to } \lceil \log_2 n \rceil \textbf{ do}$$

の部分も省略形である ($\log_2 n$ の計算は基本命令ではない).

次のプログラムは上のプログラムと同じことを, 全プロセッサを管理するスーパーバイザ (人間) の立場で書いたものである.

```
procedure sum(x[1], ..., x[n])
begin
    for i ← 1 to n do in parallel
        z[i] ← x[i]
    end;
    for h ← 1 to ⌈log₂n⌉ do
        for i ← 1 to n/2^h do in parallel
            z[i] ← z[2i − 1] + z[2i]
        end
    end;
    s ← z[1]
end
```

最初の 3 行はプロセッサ $i = 1 \sim n$ がそれぞれ $z[i] \leftarrow x[i]$ を並列に行なうことを表わしており, これは 1 単位時間で実行される. h に関する **for** 文の中の

$$\textbf{for } i \leftarrow 1 \textbf{ to } n/2^h \textbf{ do in parallel} \quad \cdots \quad \textbf{end}$$

もやはり 1 単位時間で実行されるので, このプログラムの実行時間は $\lceil \log_2 n \rceil + 2$ である.　□

6.3.2 並列アルゴリズムの技法

本書で扱ってきた並列計算のモデルは抽象的で分かりにくいかもしれないので, ここで, 具体的な問題をいくつか取り上げ, 並列化の技法を見ておこう.

プレフィックス計算

$*$ は集合 $\{x_1, x_2, \ldots, x_n\}$ の上の 2 項演算で，結合律を満たすとする．

$$s_i = x_1 * x_2 * \cdots * x_i, \quad 1 \le i \le n$$

を列 $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$ に対する**プレフックス和** (prefix sum) という．すべてのプレフィックス和 $s_1, s_2, \ldots, s_n$ を一度に求めることは逐次アルゴリズム (sequential algorithm: 通常の，並列でないアルゴリズムのこと) では $\Theta(n)$ 時間かかるが，並列アルゴリズム (PRAM アルゴリズム) では n 個のプロセッサを使って $O(\log n)$ 時間でできる．下記の prefix_sums($\mathbf{x}, n$) では，入力が大域的配列 $x[1] \sim x[n]$ に与えられるたとき，$s_1 \sim s_n$ を大域的配列 $s[1] \sim s[n]$ に求める．簡単のために，$n = 2^m$ とする．

```
procedure prefix_sums(x, n)
begin
    /* x = ⟨x_1, ..., x_n⟩ */
    for j ← 1 to n do in parallel a[0, j] ← x[j] end;
    for h ← 1 to log_2 n do
        for j ← 1 to n/2^h do in parallel
            a[h, j] ← a[h − 1, 2j − 1] * a[h − 1, 2j]
        end
    end;
    h ← log_2 n;
    while h ≥ 0 do
        for j ← 1 to n/2^h do in parallel
            if j = 1 then b[h, j] ← a[h, j] else
            if j が偶数 then b[h, j] ← b[h + 1, j/2] else
            if j > 1 が奇数 then b[h, j] ← b[h + 1, (j − 1)/2] * a[h, j]
            end end end
        end;
        h ← h − 1
    end;
    for j ← 1 to n do in parallel s[j] ← b[0, j] end
end
```

$1 \le i \le j \le n$ に対して

$$t_{ij} = x_i * x_{i+1} * \cdots * x_j$$

とする．計算は，図 6.18 に示したように，まず上向きに進行し，第 h ステップでは $t_{1+2^h(j-1),2^h j}$ を $a[h, j]$ に求める ($1 \le j \le n/2^h$)．計算が根 $a[\log_2 n, 1]$ に到達すると今度は下向きに進行し，$t_{1,2^h j}$ を $b[h, j]$ に求める ($1 \le j \le n/2^h$)．葉に到達したとき，$s_1, \ldots, s_n$ が $b[0, 1], \ldots, b[0, n]$ に得られているので，これらを $s[1], \ldots, s[n]$ に転記して終了する．この並列計算では，

$$t_{ij} = t_{ik} * t_{k+1,j} \ (i \le k < j)$$

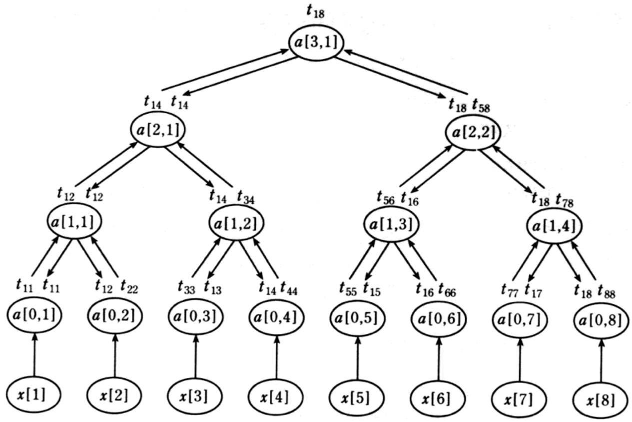


図 6.18 プレフィックス和の計算進行過程

であることを利用している．この例で用いた $a[\cdot,\cdot]$ や $b[\cdot,\cdot]$ は **2 次元配列**と呼ばれ，配列の各要素が同じサイズの配列であるような配列である ($a[i,j]$ は $a[i][j]$ の省略形 = 配列 $a[i]$ の第 j 要素)．

リストランキング (ポインタージャンプ)

リスト $L = \langle x_1, x_2, \ldots, x_n \rangle$ の要素が，図 6.19 のようにポインタで結ばれたものを**線形リスト** (linear list) という．**ポインタ** (pointer) とは，“場所を指し示す” ものである．ポインタが指す先は，図に示したように矢印 → で表わす．配列はどの要素にも 1 単位時間でアクセスできるようなデータ構造であるのに対して，線形リストにおいて要素 x_i へアクセスするためには，先頭要素 x_1 から順に $x_1 \to x_2 \to \cdots \to x_i$ とポインタを順繰りに手繰ってアクセスしなければならない．

図 6.19 の線形リストが与えられたとき，各要素のランクを求める問題を**リストランキング問題** (list ranking problem) という．ただし，要素 x_i の**ランク**とは，x_i より後ろにある要素の個数のことである (したがって，x_i のランクは

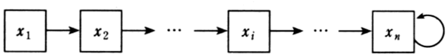


図 6.19 線形リスト

$n-i$ である).

次に示す手続き list_ranking(L) は，この問題を解くための並列アルゴリズムである．このアルゴリズムは n 個のプロセッサを用い，$O(\log n)$ 時間で実行を完了する．

```
procedure list_ranking(L)
begin
    for i ← 1 to n do in parallel
        if next[i] = next[next[i]] then rank[i] ← 0 else rank[i] ← 1 end
    end;
    for i ← 1 to n do in parallel
        while next[i] ≠ next[next[i]] do
            rank[i] ← rank[i] + rank[next[i]];
            next[i] ← next[next[i]]
        end
    end
end
```

next[i] は，要素 x_i が指す先の要素の添字を表わす．線形リストの末尾の要素は，指す先が自分自身であること (next[n] $= n$) によって表わされている．list_ranking の実行前 (第 0 ステップ) では，x_n 以外の各 x_i は 1 つ先の要素を指している (next[i] $= i+1,\ 1 \leq i < n$) が，ステップが 1 つ進むたびに，各プロセッサ i が

$$\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$$

という命令を実行してポインタを付け替え，各 x_i から 2 つ先，4 つ先，8 つ先，… の要素へのポインタが順次得られる (図 6.20 参照)．この操作を**ダブリング** (doubling) とか，**ポインタジャンプ** (pointer jumping) という．x_i のランクは rank[i] に求められる．rank を求め終わった要素は，指す先が末尾要素であること (next[i] = next[next[i]]) によって判定される．next と rank は大域的配列である．

このアルゴリズムでは，各ステップで同時に複数のプロセッサが線形リスト末尾の rank[n] を参照 (読み出し) していることに注意する．しかし，共有メモリのどこにおいても，複数のプロセッサが同時に書き込みを行なうという事態は生じていない．このような PRAM (並列アルゴリズム) は **CREW** (concurrent read exclusive write：同時読み込み排他的書き出し) であるという．同様に，**CRCW** (同時読み込み同時書き出し)，**EREW** (排他的読み込み排他的書き出し) などが定義される．同時書き込みによる競合の解消法については，演習問題 11 を参照せよ．現在，ハードウェア的に実現されているのは EREW 型の回路だけである．

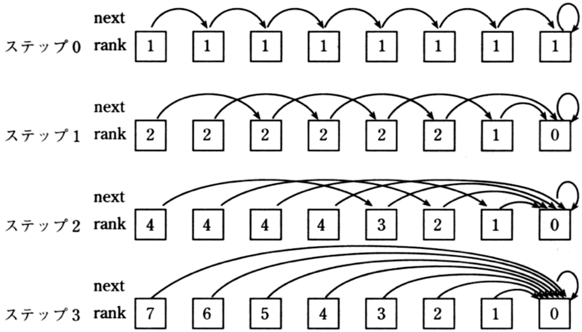


図 6.20 ダブリング

問 6.11 リストランキング問題を解く，n プロセッサ・$O(\log n)$ 時間の EREW アルゴリズムを考えよ．もっと工夫すれば，実行時間を変えずにプロセッサ数を $n/\log n$ にすることもできる．

6.3.3 PRAM と計算量

さて，すでに見てきたように，1つの PRAM プログラム (並列アルゴリズム) は何個かのプロセッサ (RAM) $P_1, \ldots, P_p$ を使って実行されるが，プログラムもプロセッサの個数 p も，入力 (＝いくつかの自然数の組) を構成する自然数の個数 m や，それらの自然数の総桁数 n に依存するのが普通である (すでに見てきた例がそうであったように，m だけに依存する場合が多い)．そこで，m, n が (文字列 1^m01^n として) 与えられたとき，必要なプロセッサの個数 p と，それら p 個のプロセッサが実行すべきプログラム (基本命令の列) を出力するような対数領域限定の変換機 (DTM) が存在するならば，その PRAM プログラムは**一様** (uniform) であるという．一様性の条件がないと，PRAM に対して意味のある計算量を定義して，それに基づいて興味ある結果を得ることができないことは，論理回路族の場合と同様である．

PRAM によって計算される関数や言語を次のように定義する．PRAM への入力は $\{0,1\}^*$ の元 $\mathbf{a} = a_1 \cdots a_n$ (各 $a_i \in \{0,1\}$) とする．入力として自然数 (の組) を扱いたい場合には，各 a_i を自然数の 2 進数表現とする．レジスタ x_0 に n の 2 進数表現をセットし，レジスタ $x_1, \ldots, x_n$ にそれぞれ $a_1, \ldots, a_n$ をセットして計算を始める．プログラムの実行が終了したときの x_0 の値を計算結果とする (言語を定義する場合には，x_0 の値が 0 か否かで受理か拒否かを決める)．長さ n の任意の入力に対し，プログラムの実行開始から停止までに実

行される基本命令の実行回数の総和が $t(n)$ 以下であるとき，その PRAM プログラムは $t(n)$ **時間限定**であるという．また，PRAM プログラムの中で使われるプロセッサの個数が $p(n)$ 以下であるとき，$p(n)$ **プロセッサ数限定**であるという．$O(t(n))$ 時間限定かつ $O(p(n))$ プロセッサ数限定の一様な PRAM プログラムによって計算される関数 (言語) のクラスを

$$\mathbf{TIME}\cdot\mathbf{PROCESSOR}(t(n), p(n))$$

で表わす．また，プロセッサ数を限定しないときは，単に

$$\mathbf{PRAM\text{-}TIME}(t(n))$$

と書く．これまでと同様に，$\mathbf{TIME}\cdot\mathbf{PROCESSOR}(t(n)^{O(1)}, 2^{O(p(n))})$ 等の記法も使う．以下のことが証明できる[††]．

命題 6.3 $\mathbf{PRAM\text{-}TIME}(t(n)^{O(1)}) = \mathbf{DSPACE}(t(n)^{O(1)})$.
したがって，次の関係も成り立つ:

$$\begin{aligned}\mathbf{PRAM\text{-}TIME}(t(n)^{O(1)}) &= \mathbf{ATIME}(t(n)^{O(1)})\\ &= \mathbf{U}_{log}\text{-}\mathbf{SIZE}\cdot\mathbf{DEPTH}(2^{t(n)^{O(1)}}, t(n)^{O(1)}).\end{aligned}$$

なお，PRAM において goto 文のジャンプ先を複数個許すことによって，非決定性の概念を導入することができる (N-PRAM: **非決定性 PRAM**)．このとき，

$$\mathbf{N\text{-}PRAM\text{-}TIME}(t(n)^{O(1)}) = \mathbf{NSPACE}(t(n)^{O(1)})$$

が成り立つ．

問 6.12 命題 6.3 の前半を仮定して，後半を証明せよ．

命題 6.4 (1) $\mathbf{NC} = \mathbf{TIME}\cdot\mathbf{PROCESSOR}((\log n)^{O(1)}, n^{O(1)})$.
(2) $\mathbf{P} = \mathbf{TIME}\cdot\mathbf{PROCESSOR}(n^{O(1)}, n^{O(1)})$.

命題 6.4 よりももっと詳細な，$\mathbf{NC}^k$ と PRAM プログラムとの間の関係が知られている (演習問題 12 参照)．命題 6.4 は，プロセッサ数が実際的な個数 (すなわち，$n^{O(1)}$ 個) に押さえられている限り，並列計算を行なっても $\mathbf{P}$ の壁を打ち破るような高速化はできないであろうこと ($\mathbf{NC} = \mathbf{P}$ は成り立たないであろうという予想にもとづく) を示している．

例 6.7 これまでに取り上げた **NC** アルゴリズムは，深さが対数多項式の論理回路によって計算するものばかりであった．上記の命題を踏まえ，ここでは，PRAM アルゴリズムにもとづく **NC** アルゴリズムの例を示しておこう．

[††] 巻末「参考書案内」の Balcázar 他の 2 章を参照せよ．

(1) 次の PRAM アルゴリズムは，n 個の要素 $x_1, \ldots, x_n$ をソートして，結果を $y[1], \ldots, y[n]$ に格納する $O(\log n)$ 時間限定かつ n^2 プロセッサ数限定の PRAM アルゴリズムである (したがって，演習問題 12 によれば，$\mathbf{NC}^2$ アルゴリズムである．演習問題 10 (2) も参照せよ)．

```
procedure sort(x_1, ..., x_n)
begin
    forall i, j (1 ≤ i ≤ n, 1 ≤ j ≤ n) do in parallel
        if x_i ≤ x_j then a[i, j] ← 1 else a[i, j] ← 0 end
    end;
    for j ← 1 to n do in parallel
        b[j] ← Σ_{i=1}^{n} a[i, j]  /* 例 6.6 の関数 sum を使う */
    end;
    for i ← 1 to n do in parallel y[b[i]] ← x[i] end
end
```

このアルゴリズムは n^2 個のプロセッサを使い，まず，第 (i, j) プロセッサが $x_i \le x_j$ かどうかを判定し，その結果を $a[i, j]$ に 0/1 で記録する．次に，第 j プロセッサが，$x_i \le x_j$ であった x_i の個数を $b[j]$ に求める (このステップで $\lceil \log_2 n \rceil + 2$ 時間かかる)．最後に，x_i は小さい方から $b[i]$ 番目の要素であるから，第 i プロセッサが $y[b[i]]$ に $x[i]$ を代入すればよい．したがって，実行に要する時間は $\lceil \log_2 n \rceil + 4$ である．

(2) 各辺に正の整数がラベル付けされた重み付きグラフ $G = (V, E)$ が与えられたとき，すべての頂点の対 (v_i, v_j) の間の，辺の重みの和が最小であるような道を求める問題 (道そのものではなく，辺の重みの和を求めればよい) を**最短経路問題** (all-pairs shortest paths problem) という．この問題が **TIME · PROCESSOR**$(\log^2 n, n^2)$ に属す (したがって，$\mathbf{NC}^3$ に属す) ことを示そう．

$V = \{v_1, \ldots, v_n\}$ とし，辺 (v_i, v_j) の重みを $w(v_i, v_j)$ で表わす．v_i から v_j へ至る，長さが h 以下の道のうちの，重みの和の最小値を $p^h(i, j)$ で表わすと，

$$p^{2h} = \min\{p^h(i, k) + p^h(k, j) \mid 1 \le k \le n\}$$

が成り立つ．このことを考慮すると，次のアルゴリズムを得る:

```
procedure shortest-weighted-path(G)
begin
    n ← |V|;
    forall i, j (1 ≤ i ≤ n, 1 ≤ j ≤ n) do in parallel p^1[i, j] ← w(v_i, v_j) end;
    for m ← 1 to ⌈log_2 n⌉ do
        h ← 2^m;
        forall i, j (1 ≤ i ≤ n, 1 ≤ j ≤ n) do in parallel
            p^h[i, j] ← min{ p^{h/2}[i, k] + p^{h/2}[k, j] | 1 ≤ k ≤ n }
        end
    end
end
```

n 個の要素の中の最小値は $O(\log n)$ 時間で並列に求められる (問 6.13) ので，このアルゴリズムの実行時間は $O(\log^2 n)$ である． □

問 6.13 n 個の要素の中の最小値を求める $\mathbf{TIME} \cdot \mathbf{PROCESSOR}(\log n, n)$ アルゴリズムを示せ．

本書で取り上げた並列計算のモデル以外にも，ベクトルマシンなど，多くのモデルが提案されている[§]． $\mathcal{F}$ をある並列計算のモデルとする．$\mathcal{F}$ に属する計算機 (アルゴリズム) によって $t(n)$ 時間以内で解くことができる問題のクラスを $\mathcal{F}(t(n))$ で表わす．PRAM を始め，多くの並列計算モデル $\mathcal{F}$ において

$$\mathcal{F}(t(n)^{O(1)}) = \mathbf{DSPACE}(t(n)^{O(1)})$$

が成り立つことが知られている．このように,「並列計算でかかる時間と逐次アルゴリズムで使う領域量とは，多項式倍の範囲では等しい」という主張は**並列計算の定立** (parallel computation thesis) と呼ばれている．

演習問題 6

1 次の言語を受理する $O(n)$ 時間限定単テープ ATM を示せ．
(a) WW^R (b) AnBn
(c) $\{x\#y \mid x, y \in \{0,1\}^*,\ y$ は $|x|$ の 2 進数表現$\}$

2 ATM に対する線形加速定理，線形領域圧縮定理を証明せよ．

3 (1) $s(n) \geq \log n$ で $s(n)$ が 領域構成可能関数ならば，$\mathbf{ASPACE}(s(n))$ は補集合で閉じていることを示せ．
(2) $s(n) = o(\log\log n)$ なら $\mathbf{ASPACE}(s(n)) = \mathbf{REG}$ である[¶]．脚注に示した論文を読み，NTM に対する類似の結果 (3 章の演習問題 3) と比較せよ．

4 補題 6.1, 6.4 の証明を再検討して次の (a), (b) を確認せよ．また，(c) を証明せよ．
(a) $s(n) \geq n$ のとき $\mathbf{NSPACE}(s(n)) \subseteq \mathbf{ATISP}(s(n)^2, s(n))$.
(b) $t(n) \geq n$ のとき $\mathbf{DTIME}(t(n)) \subseteq \mathbf{ATISP}(t(n)^2, \log t(n))$.
(c) $s(n) \geq \log n$ のとき $\mathbf{DSPACE}(s(n)) \subseteq \mathbf{ATISP}(s(n)^2, s(n))$.

5 ATM の場合には $t(n) = o(n)$ なる時間量を考えることにも意味がある．このような時間量を扱うために，次のような特殊な ATM を考える[*]．

[§]巻末「参考書案内」の Balćazar 他の 2 章を参照せよ．例えば，
V.R.Pratt and L.J.Stockmeyer, A characterization of the power of vector machines, *J. Comput. Sys. Sci.* 12, pp.198-221, 1976.
H.Alt, T.Hagerup, K.Mehlhorn and F.P.Preparata, Deterministic simulation of idealized parallel computers on more realistic ones, *SIAM J. Comput.* 16, pp.808-835, 1987.
P.W.Dymond and S.A.Cook, Complexity theory of parallel time and hardware, *Inform. Comput.* 80, pp.205-226, 1989.
[¶]K.Iwama, $\mathbf{ASPACE}(o(\log\log n))$ is regular, *SIAM J. Comput.* 22, pp.136-146, 1993.
[*]W.L.Ruzzo, On uniform circuit complexity, *J. Comput. Sys. Sci.* 22, pp.365-383, 1981.

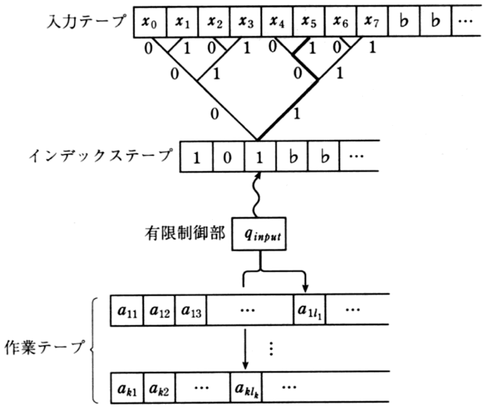


図 6.21 ランダムアクセス ATM

ランダムアクセス ATM (random access ATM) とは，インデックステープ (index tape) と呼ばれる特別なテープを持っている ATM M のことである．インデックステープには 2 進数しか書き込むことができない．動作開始時のインデックステープの内容は bb… である．入力テープは読み込み専用であり，各ます目に固定ヘッドが付いている (図 6.21 参照)．$x = x_0x_1 \cdots x_{n-1}$ を M への入力とするとき，M は特別な状態 q_{input} に入ることによりインデックステープの内容 i ($0 \leq i < |x|$) を瞬時に読み取りその値に従って入力記号 x_i を読み込む．このように，状態 q_{input} に入ったときにはインデックステープの内容 i と入力記号 x_i の読み出しは 1 ステップで行なわれるが，それ以外のときインデックステープは通常のテープとまったく同様に扱われる．ランダムアクセス ATM のことを**インデックス機能付き ATM** (indexing ATM) と呼ぶこともある．

$$\begin{aligned}\textbf{ALOGTIME} &= \textbf{ATIME}(\log n)\\ &= \{L(M) \mid M \text{ は} \log n \text{ 時間限定ランダムアクセス ATM}\}\end{aligned}$$

と定義する．**ALOGTIME** $\subseteq$ **DL** が成り立つが，逆の包含関係が成り立つか否かは現在のところ未解決である．次のことを示せ．

(1) $\mathrm{WW} = \{ww \mid w \in \{0,1\}^*\}$ や WW^R は **ALOGTIME** の元である．
(2) **REG** $\subseteq$ **ALOGTIME**.
(3) 多数決関数は **ALOGTIME** に属す．
(4) $s(n) \geq \log n$ ならば **NSPACE**$(s(n)) \subseteq$ **ATISP**$(s(n)^2, s(n))$.

6 (**DLIN =? NLIN 問題**) **DLIN** = **DTIME**(n), **NLIN** = **NTIME**(n) と定義する．次のことが知られている†．

命題 6.5 DLIN $\subsetneq$ **NLIN**.

M を ATM とする．長さ n の任意の入力 x に対し，x の計算木における根から葉に

† W.J.Paul, N.Pippenger, E.Szemerédi and W.T.Trotter, On determinism versus non-determinism and related problems, *Proc. 24th IEEE Symp. on Foundations of Comput. Sci.*, pp.429-438, 1983.

至るどの道においても，全称 ID から存在 ID (あるいは存在 ID から全称 ID) へと切り替わる回数がたかだか $a(n)-1$ 回以下であるならば，M は $a(n)$ **交代限定** (alternation bounded) であるという．k を正整数とする．k 交代限定 ATM で初期状態が存在 (全称) 状態であるものを Σ_k-**ATM** (Π_k-**ATM**) という．$t(n)$ 時間限定の Σ_k-ATM, Π_k-ATM が受理する言語のクラスをそれぞれ Σ_k-**TIME**$(t(n))$, Π_k-**TIME**$(t(n))$ で表わす．命題 6.5 の証明では次のことを使う．これを証明せよ．

(1) **DLIN** = **NLIN** と仮定すると，**DLIN** = Σ_k-**TIME**(n) がすべての自然数 k に対して成り立つ (ヒント: k に関する帰納法).

(2) **DLIN** = **NLIN** と仮定すると，**DTIME**$(t(n))$ = Σ_k-**TIME**$(t(n))$ がすべての自然数 k と任意の時間構成可能関数 $t(n)$ に対して成り立つ (ヒント: (1) の結果を詰め込み論法で拡張する).

7 (1) $\mathbf{REG} \subseteq \mathbf{NC}^1$ であることを示せ．

(2) $\{0^n1^n \mid n \geq 0\} \in \mathbf{NC}^1$ を示せ．したがって，$\mathbf{REG} \subsetneq \mathbf{NC}^1$ である．

(3) $\mathrm{MAJ}_n \in \mathbf{NC}^1$ であることを示せ．MAJ は多数決関数 (問 6.7 参照).

(4) $2\mathrm{SAT} \in \mathbf{NC}^2$ であることを示せ．

8 n ビット数 $b_n \cdots b_2 b_1$ の **2 の補数** (2's complement) とは，各ビット b_i の 0,1 を反転したものに 1 を加算して得られる 2 進数のことである．例えば，1101 の 2 の補数は 0011 である．n ビット数とその 2 の補数とを加えた値は 2^n である．2 つの n ビット数の減算 $x-y$ は，x に y の 2 の補数を加算すればよい．このことと，n ビット数の加算が $\mathbf{NC}^1$ で計算可能であることより，減算が $\mathbf{NC}^1$ で計算可能であることを示せ．

9 対数領域限定変換機によって文脈自由言語 (CFL: 2.5 節参照) に還元可能な言語のクラスを **LOGCFL** で表わす．脚注に示した論文を読んで，以下のことを確認せよ．

(1) GAP ∈ LOGCFL. 実は，$\mathbf{NC}^1 \subseteq$ LOGCFL $\subseteq \mathbf{NC}^2$ である[‡].

(2) 計算木のサイズが多項式かつ同時に対数領域限定であるような ATM によって受理される言語のクラスは LOGCFL に等しい[§].

10 (1) n 個のノードからなる 2 分木を，親子関係を表わす配列 parent, left, right で与える: parent/left/right$[i] = j \iff i$ は j の親/左の子/右の子．各ノードの深さを求める，$O(n)$ プロセッサ・$O(\log n)$ 時間の並列アルゴリズムを示せ．ヒント: 各ノードに 3 つのプロセッサ A,B,C を割り当て，左の子がいる (いない) ノードの A プロセッサは左の子の A (自分の B) プロセッサを指し，右の子がいる (いない) ノードの B プロセッサは右の子の A (自分の C) プロセッサを指し，自分が左 (右) の子である C プロセッサは親の B (C) プロセッサを指すようにし，A,B,C それぞれに初期値 $1, 0, -1$ を与える．図を描いてみよ．2 分木の底優先探索に対応する線形リストが得られる．この線形リストに対してプレフィックス和を求めよ．

(2) n 個ずつの，それぞれソートされているデータを合併して $2n$ 個のソートされたデータとする，n プロセッサ・$O(\log\log n)$ 時間 PRAM アルゴリズムが知られている．これを用いて，n 個のデータをソートする，n プロセッサ・$O(\log n \log\log n)$ 時間 PRAM アルゴリズムを示せ．最適な並列ソーティングアルゴリズムは，n プロセッサ・$O(\log n)$ 時間アルゴリズムである[¶].

11 PRAM を現実のコンピュータとして実現するためには，複数のプロセッサ

[‡] S.A.Cook, A taxonomy of problems with fast parallel algorithms, *Inform. Contr.* 64, pp.2-22, 1985.

[§] W.L.Ruzzo, Tree-size bounded alternation, *J. Comput. Sys. Sci.* 21, pp.218-235, 1980.

[¶] 巻末「参考書案内」の Jájá の 4 章を参照のこと．

$P_1, \ldots, P_k$ から同時に同じメモリへ書き込みの要求があったときに，その競合を解消する方法を決めておく必要がある．以下のような方法が考えられる:

(a) priority CRCW PRAM: プロセッサ番号が最も小さいものだけが書き込み命令を実行でき，他のプロセッサの命令は実行されない．

(b) common CRCW PRAM: $P_1, \ldots, P_k$ すべてが同じ値を書き込もうとしたときだけ書き込みが行なわれる．

(c) arbitrary CRCW PRAM: $P_1, \ldots, P_k$ の中のどれか適当な 1 つだけが書き込みに成功し，それ以外のプロセッサの命令は実行されない．ただし，このような PRAM モデルを用いる場合には，選ばれるプロセッサがどれであっても正しく動くようなアルゴリズムでないといけない．

(a), (b) および EREW PRAM の間に大きな差はないことが知られている[‖]．次の命題の (1) を証明せよ．

命題 6.6 (1) プロセッサ数 p の priority CRCW PRAM の 1 ステップを，プロセッサ数 p^2 の common CRCW PRAM により $O(1)$ ステップで模倣することができる．

(2) プロセッサ数 p の priority CRCW PRAM の 1 ステップを，プロセッサ数 p の EREW PRAM により $O(\log p)$ ステップで模倣することができる．

12 k を非負整数とする．多項式サイズかつ深さが $O(\log^k n)$ の，ファンインが無制限で対数一様な論理回路族によって受理される言語のクラスを $\mathbf{AC}^k$ で表わす．$\mathbf{AC} = \bigcup_{k \geq 0} \mathbf{AC}^k$ とする．$\mathbf{NC}^k \subseteq \mathbf{AC}^k \subseteq \mathbf{NC}^{k+1}$ であることを示せ．したがって，$\mathbf{AC} = \mathbf{NC}$ である．

$\mathbf{AC}^k = \mathbf{TIME} \cdot \mathbf{PROCESSOR}((\log n)^k, n^{O(1)})$ であることが知られている[**]．$\mathbf{NC}^1$ に属するが $\mathbf{AC}^0$ には属さない問題 (関数) としては，パリティ関数，多数決関数，乗算，ブール行列の推移閉包を求める問題などが知られている[††]．

13 1 つの**ランダムアクセス機械** (RAM) だけを考えると，その計算能力はチューリングマシンに等しいことが証明できる．そのためには，"関数" あるいは "言語" を RAM を用いて定義する必要がある．この定式化を試みよ．

14 次のことが成り立つ:

$$\begin{aligned}\mathbf{P} &= \mathbf{DTISP}(n^{O(1)}, n^{O(1)}) = \mathbf{DTIRE}(n^{O(1)}, n^{O(1)}) \\ &= \mathbf{ATISP}(n^{O(1)}, \log n) \\ &= \mathbf{U}_{log}\text{-}\mathbf{SIZE} \cdot \mathbf{DEPTH}(n^{O(1)}, n^{O(1)}) \\ &= \mathbf{TIME} \cdot \mathbf{PROCESSOR}(n^{O(1)}, n^{O(1)}).\end{aligned}$$

同様の関係を，**NC** についてもまとめよ．

[‖] L.Kučera, Parallel computation and conflicts in memory access, *Inform. Proc. Lett.* 14, pp.93-96, 1982.

U.Vishkin, Implementation of simultaneous memory address access in models that forbit it, *J. Algorithms* 4, pp.45-50, 1983.

[**] L.J.Stockmeyer and U.Vishkin, Simulation of parallel random access machines by circuits, *SIAM J. Comput.* 13, pp.409-423, 1984.

[††] M.Furst, J.Saxe and M.Sipser, Parity, circuits, and the polynomial time hierarchy, *Math. Sys. Theory* 17, pp.114-118, 1984.

7
確率性アルゴリズムと近似計算

前章までは，さまざまな計算問題を，それを解くために必要な時間量や領域量によって分類し，ある種の問題に対しては効率の良いアルゴリズムが存在しない (あるいは存在しないであろうと予想される) ことを述べた．例えば NP 完全といわれる問題がその例である．こういった問題については，効率良いアルゴリズムの設計はあきらめるしかないのであろうか？ この章ではこの疑問に 2 つの観点から取り組む．

つねに 100 %正しい答を出すわけではないが，計算の効率が高く，しかも間違った答を出す確率がきわめて小さいアルゴリズムであれば，実用上は十分役立つ．この章で最初に考察するのは，このような性質を持つ“確率性アルゴリズム”である．確率性アルゴリズムの基本的性質や，確率性アルゴリズムによって解ける問題にはどのようなものがあるかといったことなどを考察することによって，確率性アルゴリズムの有効性や限界について論じる．

また，計算が困難と思われる問題の中には，厳密解を求めるのは確かに難しいが近似解ならば効率良く計算できるものがある．本章の後半ではそのような“近似アルゴリズム”について考察する．

7.1 確率性計算のモデル

この節では，まず確率性アルゴリズムの具体例とその有効性について見た上で，確率性計算のさまざまな形式的モデルを導入し，それらの間の関係について考察する．

7.1.1 確率性アルゴリズムとは

確率性アルゴリズム (probabilistic algorithm) とは，計算の途中でコインを

投げて，表が出るか裏が出るかで次に実行するステップを二者択一的に選ぶようなアルゴリズムである．その結果得られた答は正しいことも誤っていることもある．しかし，起こりうる場合すべてのうち，誤った答に至る場合がきわめて少なければ (すなわち，誤った答を出す確率がごく小さいなら)，このようなアルゴリズムでも十分実用に役立つ．まず，そのようなアルゴリズムの具体例として，Rabin による素数判定のアルゴリズムを考えてみよう*.

定理 5.8 (5.2.2 節) で見たように，n が合成数ならば，フェルマーテスト

$$a^{n-1} \equiv 1 \pmod{n}?$$

に対する答が no となるような a $(0 < a < n)$ が必ず存在する．もし，そのような a の個数が，どんな n についても，可能な候補すべてのうちの半分 ($= \frac{n-1}{2}$ 個) 以上あると仮定すると，次のような確率性アルゴリズムによって，n が合成数であるか否かを非常に高い確率で判定することができる．k を適当な正整数とする．

```
function Rabin's_primality_test(n) /* first version */
begin
    for i ← 1 to k do /* 以下のことを k 回行なう */
        0 < a < n なる a をランダムに選ぶ;
        if a^{n-1} ≢ 1 (mod n) then
            return "n は合成数である"
        else
            return "多分，n は素数であろう"
        end
    end
end
```

このアルゴリズムによって“合成数である (素数でない)”と判定された n は確かに合成数である．また，上述の仮定が正しいとすると，“多分，素数であろう”という判定が誤っている確率は $(\frac{1}{2})^k$ 未満である．k が大きければ，これは十分小さい誤り率である．しかしながら，次の問に述べるように，上述の仮定は実は正しくない．

問 7.1 $\Phi(n) = \{a \mid 0<a<n,\ \gcd(a,n) = 1\}$ と定義する．$0<a<n$, $a \notin \Phi(n)$ なら $a^{n-1} \not\equiv 1 \pmod{n}$ である (系 5.3 参照)．したがって，$a \notin \Phi(n)$ なる任意の a は，n が合成数であることの“証拠”である．このことを使って，$n = 561, 1105, 1729$ のどれも上述の仮定を満たさないことを示せ．これらの数のように，$\Phi(n) = \{a \mid 0<a<n,\ a^{n-1} \equiv 1 \pmod{n}\}$ を満たす合成数 n のことを **Carmichael 数**という．

上述の仮定は成り立たないものの，それに匹敵する次の命題が成り立つ．証

*M.O.Rabin, Probabilistic algorithms, in J.Traub ed. *Algorithms and Complexity: New Direction and Results*, Academic Press, pp.21-39, 1976. この論文は，その後の確率性アルゴリズムの理論的研究の魁となった．

明は他書に譲る[†]が，これに基づいて素数性を判定する多項式時間の確率性アルゴリズムを作ることができる (図 7.1)．このアルゴリズムを k 回反復することにより，誤った答を出力する確率を $(\frac{1}{2})^k$ より小さくすることができる．

命題 7.1 n が奇数の合成数ならば，次の条件を満たす整数 $0 < a < n$ が少なくとも $(n-1)/2$ 個存在する:
$a^{n-1} \not\equiv 1 \pmod{n}$ が成り立つか，または，
$$2^i \mid (n-1),\ 1 < \gcd(a^{(n-1)/2^i} - 1, n) < n$$
を満たす整数 i が存在する．

逆に，この命題の条件が成り立つなら n は合成数である (問: なぜか？)．このような a のことを，n が合成数であることの**証拠** (witness) と呼ぶ．

```
function primality_test(n) /* Rabin's algorithm: second version */
begin
    if n が偶数 then return "n は合成数である";
    0 < a < n なる整数 a をランダムに求める;
    if a は n が合成数であることの証拠である then
        return "n は合成数である"
    else
        return "多分，n は素数であろう"
    end
end
```

図 7.1 素数かどうかを判定する確率性アルゴリズム

5.2.2 節で述べたように，合成数かどうかの判定問題は **P** に属すかどうかまだ分かっていないが，この primality_test(·) はそれに対する多項式時間確率性アルゴリズムである．一方，たとえ **P** に属す問題であっても，その問題を解く確率性アルゴリズムが，既知のアルゴリズムよりも高速であるとか，あるいは高速にならないまでも決定性アルゴリズムよりもシンプルであるならば，確率性であることを差し引いても十分有用である場合もある．そのような例については，演習問題 2 を参照せよ．

7.1.2 確率性 TM

確率性アルゴリズムとは何かを形式的に定義するために，確率性チューリングマシンという概念を導入しよう[‡]．$t(n)$ を関数 $\mathbf{N} \to \mathbf{N}$ とする．次の条件を

[†] 巻末参考書: Bovet–Crescenzi の 9 章を参照せよ．類似の結果 (R.Solovay–V.Strassen による素数判定の確率性アルゴリズム) については，Papadimitoriou の 11 章を参照せよ．

[‡] J.Gill, Computational complexity of probabilistic Turing machines, *SIAM J. Comput.* 6, pp.675-695, 1977. PP, BPP, RP, ZPP といったクラスをはじめ，確率性 TM に関する基本的な概念のほとんどはこの論文で導入された．

満たす NTM M のことを $t(n)$ 時間限定の**確率性 TM** (**PTM**: probabilistic TM) という:

- M は各ステップで必ずちょうど 2 つの非決定性選択肢を持つ．これは，各ステップごとに M はコインを投げて，表が出たか裏が出たかに応じて 2 つの分岐のどちらを選ぶかをランダムに決めることを表わしている．
- 入力 x に対するどの計算もすべて同じステップ数 $t(|x|)$ で終了し，しかも終了時の状態は必ず受理状態か拒否状態である．

これらの条件より，PTM の計算木は高さが $t(|x|)$ の完全 2 分木 (葉以外のどのノードも子をちょうど 2 つ持ち，かつどの葉の深さも同じであるような 2 分木) であり，どの葉においても，そこに対応する ID は受理 ID か拒否 ID である．$2^{t(|x|)}$ 個の葉のうち，受理 ID が対応しているものが半数より多い ($> 2^{t(|x|)-1}$) とき，M は x を**受理**し，そうでないとき**拒否**すると定義する．ただし，ほとんどの文献でそうであり，本書でもそうであるが，「M が受理 ID に入る」ことを「M が x を受理する」ということが多い (M を NTM と見ているため) ので注意したい．

はじめに，PTM の計算能力がどの程度のものであるかを述べておこう．

定理 7.1 $t(n) \geq n$ を領域構成可能関数とし，M を $t(n)$ 時間限定の PTM とする．M を模倣する，$O(c^{t(n)})$ 時間限定かつ $O(t(n))$ 領域限定の DTM M' が存在する．

証明 M' を次のように定義する．

```
procedure M'(x)
begin
    t(|x|) を生成する;
    カウンターを 0 にする;
    forall y ∈ {0,1}* s.t. |y| = t(|x|) do
        入力 x に対する M の動作を 1 ステップずつ模倣する;
        ただし，計算木の深さ d の頂点における分岐は
            if y の d ビット目 = 0 then 左へ分岐 else 右へ分岐 end
        とする;
        if M が受理 ID に入った then カウンタに 1 加える end
        /* このような場合，if M が x を受理 then ··· ともいう */
    end;
    if カウンターの値 > 2^{t(|x|)-1} then x を受理する else 拒否する end
end
```

M' は M の可能な計算すべてを模倣し，それらのうち M が入力を受理した場合を数えてカウンタに記録する．$|x| = n$ とする．$t(n)$ は領域構成可能関

数であるから，M' は $t(n)$ を $O(c_1^{t(n)})$ 時間で生成することができ (定理 3.8 および定理 4.1 (2) 参照)，さらに，$y \in \{0,1\}^*$ に従って M の計算木の 1 つの道をたどるのに必要な時間は $O(|y|) = O(t(n))$ であり，それを $2^{t(n)}$ 回行なうから，M' が使う時間の総計は $O(t(n)2^{t(n)}) = O(c_2^{t(n)})$ である (c_1, c_2 は適当な正定数).

一方，M' が使う領域量は「M が使う領域量」+「y を保持するのに必要な領域量 $(= O(|y|))$」+「カウンタに使うための領域量」であり，それは $O(t(|x|))$ である．　□

この定理は，同じ問題を解くのなら，一般には PTM (確率性アルゴリズム) を使った方が DTM (決定性アルゴリズム) を使うよりずっと高速に計算できるということを示している (とはいえ，PTM だと $t(n)$ 時間で計算できるのに DTM だとどうしても $c^{t(n)}$ 時間かかってしまうといった問題の例は，一つも知られていない)．しかし，確率性アルゴリズムを使う場合に問題になるのは，計算結果の信頼性である．そこで，入力 x に対する PTM M による計算の**エラー確率** (error probability) $\mathit{error}_M(x)$ を次のように定義する．x に対する M の計算の総数 (すなわち，計算木の根から葉へ至る道の個数) を $\#\mathit{total}_M(x)$ とし，そのうちの受理計算の個数を $\#\mathit{accept}_M(x)$，拒否計算の個数を $\#\mathit{reject}_M(x)$ とする:

$$\mathit{error}_M(x) = \begin{cases} \#\mathit{reject}_M(x)/\#\mathit{total}_M(x) & M \text{ が } x \text{ を受理したとき} \\ \#\mathit{accept}_M(x)/\#\mathit{total}_M(x) & M \text{ が } x \text{ を拒否したとき.} \end{cases}$$

定義より，つねに $0 \le \mathit{error}_M(x) \le \frac{1}{2}$ である．

以下では，多項式時間限定の PTM だけを考える．多項式時間限定 PTM のことを **PP-TM** と呼ぶ (PP は probabilistic polynomial time の頭文字)．すなわち，M が PP-TM であるとき，

- $\#\mathit{accept}_M(x)/\#\mathit{total}_M(x) > \frac{1}{2}$ なら M は x を受理し，
- $\#\mathit{reject}_M(x)/\#\mathit{total}_M(x) \ge \frac{1}{2}$ なら M は x を拒否する．

PP-TM が受理する言語のクラスを **PP** で表わす．

例 7.1 ブール式 $\psi(x_1, \ldots, x_n)$ が与えられたとき，ψ に対する付値 (ブール変数 $x_1, \ldots, x_n$ への値の割り当て方) は全部で 2^n 通りあるが，そのうちの半数より多くが ψ を充足するかどうかを決定する問題を **MAJSAT** と呼ぶ．この問題が **PP** に属すこと，すなわち，

$$L_{MAJSAT} = \{\text{ブール式 } \psi \text{ のコード} \mid \psi \text{ の付値のうち，半数より多くが } \psi \text{ を充足させる}\}$$

```
procedure M(ψ)
/* ψ はブール式のコード */
begin
    ψ がブール式の形をしているか否かチェックする;
    ψ の変数を x_1, ..., x_n とする;
    for i ← 1 to n do
        x_i ← random (0,1);
        ψ ← (ψ に x_i を代入して簡単化したブール式);
        if ψ = 1 then 入力を受理して停止 else 拒否して停止 end
    end
end
```

図 **7.2** L_{MAJSAT} を受理する PP-TM

と定義したとき，言語 L_{MAJSAT} を受理する PP-TM M が存在することを示そう (図 7.2 参照)．M は ψ の変数 $x_1, \ldots, x_n$ に順次 0 か 1 をランダムに割り当てながら計算を進める．**random** (0,1) は 0 か 1 をそれぞれ確率 $\frac{1}{2}$ で生成する**乱数生成関数**である (定義上は，PTM の動作はこのような理想的な乱数にもとづいて行われるわけであるが，実際にそのような真の乱数を作ることは不可能である．このことについては 7.1.3 節で考察する)．また，図 7.2 では M を PTM らしく記述しているが，本来の定義によると PTM は NTM であるから，この部分では $\delta(q, \cdot) = \{(q_L, \cdot, \cdot), (q_R, \cdot, \cdot)\}$ といった非決定性の動作が行なわれる ($\cdot$ は適当な記号)．受理/拒否計算に対応する割り当ては ψ を充足する/しない．M が多項式時間限定であることは明らかであろう． □

PP は，これから導入するいくつかの確率性計算量のクラスの中でもっとも制約が少ないクラスである．計算量の階層における **PP** の位置を次の定理に述べる．

定理 7.2 **NP** $\subseteq$ **PP** $\subseteq$ **PSPACE**.

証明 定理 7.1 より，**PP** $\subseteq$ **PSPACE** である．

NP $\subseteq$ **PP** を示すために，$L \in$ **NP** とする．L を受理する多項式時間限定 NTM を M とする．次のように動作する PTM M' を考える: M' は M の初期状態とは異なる別の初期状態 s'_0 を持ち，s'_0 から非決定性の動作で 2 分岐する．そのうちの 1 つは M の初期状態に入って，そのまま M の動作を模倣し (受理・拒否は M のそれに従う)，他の 1 つは M と同じステップ数を実行した後に必ず入力を受理する．

明らかに，M は多項式時間限定であり，

$$
\begin{aligned}
x \in L(M) &\iff x \text{ に対する } M' \text{ の計算の半数より多くが受理計算である} \\
&\iff x \in L(M')
\end{aligned}
$$

が成り立つ．よって，$L \in \mathbf{PP}$ である． □

問 7.2 (1) 任意の PP-TM M に対し，M と同じ言語を受理する PP-TM M' で，任意の入力 x に対して必ず $\#accept_{M'}(x)/\#total_{M'}(x) \neq 1/2$ であるようなものが存在することを示せ．ヒント: M が $p(n)$ 時間限定のとき，M' は M を $p(n)$ ステップ模倣した後，さらに $p(n)$ ステップをコイン投げだけをしながら実行する．M が入力を受理し，かつコイン投げで 1 回以上表が出たときだけ，M' は $2p(n)$ ステップ目に入力を受理する．

(2) $\mathbf{PP}$ は補集合で閉じていることを示せ．したがって，$\mathbf{NP} \cup \text{co–}\mathbf{NP} \subseteq \mathbf{PP}$ である．

(3) 実は，$\mathbf{PP}$ は $\cup$ や $\cap$ でも閉じている (演習問題 5 参照)．これらの証明に比べ，$\mathbf{PP}$ が対称差で閉じていること ($L_1, L_2 \in \mathbf{PP} \Longrightarrow (L_1 - L_2) \cup (L_2 - L_1) \in \mathbf{PP}$) を示すのはむずかしいことではない．ヒント: 2 つの PP-TM を模倣し，そのうちのどちらか片方だけが受理した入力だけを受理する．

#SAT についてはすでに数え上げ問題として 5.5 節で述べたが，ここでは改めて言語の形で定義しよう (本来の #SAT は言語 (決定問題) ではなく，語から自然数への関数 (関数問題) である):

$$\#\text{SAT} = \{\langle \psi, m \rangle \text{ のコード} \mid \psi \text{ はブール式，} m \text{ は自然数，} \psi \text{ は } m \text{ 個より多くの付値によって充足される}\}.$$

以下で，#SAT と MAJSAT が $\mathbf{PP}$ 完全な問題であることを示す．次の補題の証明から始める．

補題 7.1 $\mathbf{PP}$ は多項式時間還元 $\leq_{poly}$ で閉じている．すなわち，$L' \leq_{poly} L$ で $L \in \mathbf{PP}$ なら $L' \in \mathbf{PP}$ である．

証明 $L \in \mathbf{PP}$, $L' \leq_{poly} L$ via M とする．DTM M は $p(n)$ 時間限定であるとし，L を受理する $q(n)$ 時間限定 PTM を N とする ($p(n), q(n)$ はともに多項式)．

次のように動作する PTM M' を考える．入力 x ($|x| = n$) が与えられたとき，M' はまず 2 分岐してそれぞれの分岐先で M の模倣を開始する．それぞれの分岐先で M' が行なう計算はまったく同じで，$M(x)$ を出力テープに書き出して $p(n)$ ステップ後に模倣を終了する．つづいて，M' は $M(x)$ を入力として N の動作を模倣し，N が $M(x)$ を受理したら M' は入力 x を受理して計算を終了する．この後半の模倣にかかる時間は $O(q(p(n)))$ であるから，M' が使う時間は，M を模倣する前半と合わせて $O(p(n) + q(p(n)))$ である．一方，$x \in L' \Longleftrightarrow M(x) \in L \Longleftrightarrow N$ が $M(x)$ を受理する $\Longleftrightarrow M'$ が x を受理する，が成り立つから，M' は L' を受理する．よって，$L' \in \mathbf{PP}$． □

補題 7.2 #SAT は $\mathbf{PP}$ 困難である．

証明 $L \in \mathbf{PP}$ とし，L を受理する $p(n)$ 時間限定 PP-TM を M とする．M が入力 x を受理する必要十分条件は，M の受理計算の個数 $> 2^{p(n)-1}$ である．

ブール式 E_x を，定理 5.4 (Cook の定理) の証明において構成したのと同様に定義する．E_x の構成の仕方より，入力 x に対する M の受理計算と E_x を充足させる付値とが 1 対 1 に対応する．よって，M が x を受理する必要十分条件は E_x を充足させる付値の個数 $> 2^{p(n)-1}$ である．x を入力したとき $\langle E_x, 2^{p(|x|)} - 1\rangle$ のコードを出力する，多項式時間限定変換機 N を構成することは容易である (コードの一部である $2^{p(|x|)-1}$ は 2 進数で表わされる)．すなわち，$L \leq_{poly}$ #SAT via N.　□

定理 7.3 #SAT および MAJSAT は **PP** 完全である．

証明 例 7.1 より，MAJSAT $\in \mathbf{PP}$ である．よって，補題 7.1 と補題 7.2 より，#SAT $\leq_{poly}$ MAJSAT であることを示せばよい．$x = \langle \psi, m\rangle$ を #SAT のインスタンスとする (ψ は $y_1, \ldots, y_n$ を変数とするブール式，$m \leq 2^n$ は自然数)．$m = 2^{n-r_1} + 2^{n-r_2} + \cdots + 2^{n-r_k}$, $1 \leq r_1 < r_2 < \ldots < r_k \leq n$ とする．このとき，

$$
\begin{aligned}
\varphi_m \quad = \quad & (y_1 \wedge \ldots \wedge y_{r_1}) \vee \\
& (\neg y_1 \wedge \ldots \wedge \neg y_{r_1} \wedge y_{r_1+1} \wedge \ldots \wedge y_{r_2}) \vee \\
& (\neg y_1 \wedge \ldots \wedge \neg y_{r_2} \wedge y_{r_2+1} \wedge \ldots \wedge y_{r_3}) \vee \\
& \cdots \qquad\qquad\qquad\qquad\qquad\qquad \cdots \vee \\
& (\neg y_1 \wedge \ldots \wedge \neg y_{r_{k-1}} \wedge y_{r_{k-1}+1} \wedge \ldots \wedge y_{r_k})
\end{aligned}
$$

と定義する．φ_m の i $(1 \leq i \leq k)$ 番目の “リテラルの積の部分” $(\cdots \wedge y_{r_i})$ を充足させる付値の個数はちょうど 2^{n-r_i} 個である．しかも，それらは他のどの “リテラルの積の部分” を充足させることもない．したがって，φ_m を充足させる付値の個数はちょうど $2^{n-r_1} + 2^{n-r_2} + \cdots + 2^{n-r_k} = m$ 個であり，$\neg\varphi_m$ を充足させる付値の個数は $2^n - m$ 個である．y を新しいブール変数とし，

$$\phi_x = (\psi \wedge y) \vee (\neg\varphi_m \wedge \neg y)$$

と定義すると，ψ を充足させる付値の個数が m より大きい必要十分条件は，ϕ_x の付値のうち半数 ($= 2^n$ 個) より多くが ϕ_x を充足させることである．入力 x に対して ϕ_x を生成する，多項式時間限定の変換機 N を構成することは容易である．よって，#SAT $\leq_{poly}$ MAJSAT via N である．　□

7.1.3　エラー確率限定 PTM：クラス BPP と RP

$p(n)$ 時間限定 PP-TM においては，エラー確率は最悪の場合には $\frac{1}{2}$ になる (受理計算の個数 = 拒否計算の個数のとき)．エラー確率が 0 に近いほど信頼性

が高い確率性アルゴリズムであることは言うまでもない．そこで，任意の入力に対するエラー確率が，$\frac{1}{2}$ より真に小さい定数で押さえられているようなPP-TMを **BPP-TM** (bounded error probabilistic polynomial time TM) という．すなわち，M がBPP-TMであるとは，ある定数 $0 < \epsilon < \frac{1}{2}$ が存在して，

- 任意の x に対して
$$\#accept_M(x)/\#total_M(x) > \tfrac{1}{2} + \epsilon \text{ または}$$
$$\#reject_M(x)/\#total_M(x) > \tfrac{1}{2} + \epsilon$$
が成り立ち，かつ
- $\#accept_M(x)/\#total_M(x) > \frac{1}{2} + \epsilon$ なら M は x を受理し，
- $\#reject_M(x)/\#total_M(x) > \frac{1}{2} + \epsilon$ なら M は x を拒否する

が成り立つことである．

BPP-TMが受理する言語のクラスを **BPP** で表わす．定義から明らかに **BPP** $\subseteq$ **PP** である．

7.1.1節で述べた素数判定アルゴリズム primality_test(·) はBPPアルゴリズムであり (実は，後述するZPPアルゴリズム (=ラスベガスアルゴリズム) でもある)，繰り返し実行することによってエラー確率をいくらでも小さくすることができた．次の定理に述べるように，このことはBPPアルゴリズム一般に成り立つ基本的な性質である:

定理 7.4 $L \in$ **BPP** である必要十分条件は，任意の多項式 $p(n)$ に対し，L を受理する $p(n)$ 時間限定かつエラー確率がたかだか $(\frac{1}{2})^{p(n)}$ のBPP-TMが存在することである．

証明 ($\Longleftarrow$) は自明である．

($\Longrightarrow$) を証明しよう．L を受理するBPP-TMを M とし，M のエラー確率を $0 < \epsilon < \frac{1}{2}$ とする．次のPTM M' を考える．m は整数であるが，その値は後で決める．M' は M の模倣を m 回繰り返す．

```
procedure M'(x)
begin
    counter ← 0;
    for i ← 1 to m do
        x を入力として M を模倣する;
        if M が x を受理 then counter ← counter + 1 end
    end;
    if counter > m/2 then x を受理する else 拒否する end
end
```

最初に M が x を受理する場合を考える．入力 x に対する M' の計算のうちcounterの値がちょうど $i \leq m/2$ であるようなものの個数を $c(x,i)$ とすると，

$$\begin{aligned} c(x,i)/\#total_{M'}(x) &\le \binom{m}{i}\left(\frac{1}{2}+\epsilon\right)^i\left(\frac{1}{2}-\epsilon\right)^{m-i} \\ &\le \binom{m}{i}\left(\frac{1}{2}+\epsilon\right)^i\left(\frac{1}{2}-\epsilon\right)^{m-i}\left(\frac{\frac{1}{2}+\epsilon}{\frac{1}{2}-\epsilon}\right)^{m/2-i} \\ &= \binom{m}{i}\left(\frac{1}{4}-\epsilon^2\right)^{m/2} \end{aligned}$$

である．したがって，

$$\begin{aligned} \#reject_{M'}(x)/\#total_{M'}(x) &\le \sum_{i=0}^{(m-1)/2}\binom{m}{i}\left(\frac{1}{2}+\epsilon\right)^i\left(\frac{1}{2}-\epsilon\right)^{m-i} \\ &\le \sum_{i=0}^{(m-1)/2}\binom{m}{i}\left(\frac{1}{4}-\epsilon^2\right)^{m/2} \\ &= 2^{m-1}\left(\frac{1}{4}-\epsilon^2\right)^{m/2} \\ &= \frac{1}{2}\left(1-4\epsilon^2\right)^{m/2} \end{aligned}$$

である．定理が成り立つためには，

$$\tfrac{1}{2}\left(1-4\epsilon^2\right)^{m/2} \le \left(\tfrac{1}{2}\right)^{p(n)}$$

が成り立てばよいが，そのためには $m \ge 2(p(n)-1)/\log_2\left(1/(1-4\epsilon^2)\right)$ であればよい．

M が x を拒否する場合についても同様に証明できる．

このように，M を m 回繰り返して模倣する (それぞれは互いに独立の模倣である) ことにより，M' のエラー確率を望み通り小さくすることができる． □

BPP アルゴリズムは理論的にも実用的にも重要なものであるが，このアルゴリズムが理論通り働くためには "完璧な 0/1 乱数" が必要である．無限ビット列 $(x_1, x_2, \ldots)$ が**完璧な 0/1 乱数** (perfect random source) であるとは，任意の $n > 0$ と任意の $(y_1, \ldots, y_n) \in \{0,1\}^n$ に対して

$$\mathrm{Prob}[x_i = y_i \ (i = 1, \ldots, n)] = 2^{-n}$$

が成り立つことである．すなわち，どの x_i も独立で，$x_i = 1$ である確率が $\frac{1}{2}$ である．しかし，現実には完璧な 0/1 乱数を生成する術はない (せいぜい，乱数らしきものしか生成できない)．そこで，完璧な 0/1 乱数に近いものとして次のようなものを考える．

$0 \le \delta \le \frac{1}{2}$ とし，γ を $\{0,1\}^*$ から閉区間 $[\delta, 1-\delta]$ への適当な関数とする (γ として，"完璧な 0/1 乱数に近いものを生成する関数" を念頭に置いている．したがって，γ は非常に複雑な関数であるはずである．3 章の演習問題 11 を参照せよ)．最初の n ビットが $y_1, \ldots, y_n$ である確率が

$$\prod_{i=1}^{n}(y_i \cdot \gamma(y_1 \cdots y_{i-1}) + (1-y_i)(1-\gamma(y_1 \cdots y_{i-1}))$$

であるような無限ビット列を δ **乱数** (δ-random source) と呼ぶ．この式は，i 番目のビットが 1 である確率が $\delta \le \gamma(y_1 \cdots y_{i-1}) \le 1-\delta$ であることを示している．$\frac{1}{2}$ 乱数は完璧な 0/1 乱数である．$0 < \delta < \frac{1}{2}$ なる δ 乱数のことを**乱数もどき** (slightly random source) と呼ぶ．

乱数もどきであっても実用的には十分役立つことを理論的に示すために，δ 乱数に従ってランダムな遷移を行なう PTM を形式的に定義しよう．

M を $p(n)$ 時間限定の PTM とする ($p(n)$ は多項式)．$m = p(n)$ とおく．長さ n の入力 x に対し，M の計算木は高さが m の完全 2 分木で，2^m 個の葉と $2^{m+1}-2$ 個の辺を持つ．この計算木を $\mathcal{T}_{M,x}$ で表わす．

$0 \le \delta \le \frac{1}{2}$ とする．$\mathcal{T}_{M,x}$ に対する δ **割り当て** (δ-assignment) とは，$\mathcal{T}_{M,x}$ の辺の集合 $E(\mathcal{T}_{M,x})$ から区間 $[\delta, 1-\delta]$ への関数 γ で，e_L, e_R が同じノードから出てそれぞれその左右の子へ向かう辺であるならば，$\gamma(e_L) + \gamma(e_R) = 1$ を満たしているようなもののことである．図 7.3 に 0.1 割り当ての例を示した．

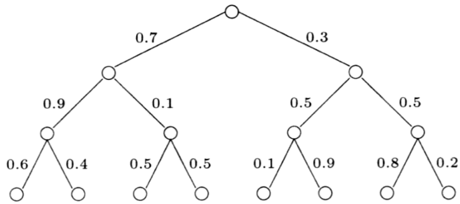


図 7.3　計算木への 0.1 割り当て

δ 割り当て γ が与えられたとき，$\mathcal{T}_{M,x}$ の葉 ℓ における "受理確率" を

$$\mathrm{Prob}[\ell] = \textstyle\prod_{e \in path(\ell)} \gamma(e)$$

で定義する．ここに，$path(\ell)$ は $\mathcal{T}_{M,x}$ の根から ℓ へ至る道 (を構成する辺の集合) である．$\mathrm{Prob}(\ell)$ は道 $path(\ell)$ が選ばれる確率を表わしている．

$$\mathrm{Prob}[M \text{ が } x \text{ を受理する}] = \textstyle\sum_{\ell \text{ は受理 ID を持つ葉}} \mathrm{Prob}[\ell]$$

と定義する．言語のクラス δ-**BPP** を，

$$L \in \delta\text{-}\mathbf{BPP} \overset{\text{def}}{\iff} \begin{array}{l}\text{次の条件を満たす PTM } M \text{ が存在する：}\\ x \in L \iff \mathrm{Prob}[M \text{ が } x \text{ を受理する}] \ge \frac{3}{4}\end{array}$$

により定義する．$\frac{3}{4}$ に特に深い意味はない (定理 7.4 参照)．0-**BPP** = **P**，$\frac{1}{2}$-

BPP = **BPP** である．一般に，次の命題が成り立つ*．これは，乱数もどきでも十分 BPP アルゴリズムのための乱数として役立ちうることを示している．

命題 7.2 任意の $0 < \delta \leq \frac{1}{2}$ に対して，δ-**BPP** = **BPP** である．

7.1.4 モンテカルロアルゴリズムとラスベガスアルゴリズム

正しい答は "no" なのに誤って "yes" と答えることが決してない (したがって，"no" と出力した答は誤っていることもあるが，"yes" と出力した答には誤りがない) ような多項式時間の確率性アルゴリズムのことを**モンテカルロアルゴリズム** (Monte Carlo algorithm) という．これに対応する PTM を **RP-TM** (RP は randomized polynomial time の頭文字) という．すなわち，RP-TM とは，

- 任意の x に対して
  $\#accept_M(x)/\#total_M(x) > \frac{1}{2}$ または $\#reject_M(x)/\#total_M(x) = 1$
  が成り立ち，かつ
- $\#accept_M(x)/\#total_M(x) > \frac{1}{2}$ ならば M は x を受理し，
- $\#reject_M(x)/\#total_M(x) = 1$ ならば M は x を拒否する

が成り立つような PP-TM M のことである．

RP-TM が受理する言語のクラスを **RP** で表わす．すでに 7.1.1 節で見たように，PRIME $\in$ co-**RP** である (実は，PRIME $\in$ **RP** であることも知られており†，素数性判定問題は後述するクラス **ZPP** に属す)．

問 7.3 7.1.1 節の素数判定アルゴリズム primality_test(·) に基づいて，合成数の集合 PRIME^c を受理する RP-TM を示せ．

定理 7.5 **RP** $\subseteq$ **NP** $\cap$ **BPP**.

証明 $L \in$ **RP** を受理する RP-TM を M とする．定理 7.4 の証明と同様に M を 2 回模倣し，入力 x に対する M の計算のうち少なくとも 1 つが受理計算だったら x を受理するように定義した PTM を M' とする．このとき，もし M が x を受理するなら $\#reject_M(x)/\#total_M(x) < \frac{1}{2}$ であるから $\#accept_{M'}(x)/\#total_{M'}(x) > 1 - \frac{1}{4}$ であり，M が x を拒否するなら

---

*U.V.Vazirani and V.V.Vazirani, Random polynomial time equals semi-random polynomial time, *Proc. 26th IEEE Symp. on the Foundations of Comput. Sci.*, pp.417-428, 1985.
D.Zuckerman, Simulation of BPP using a general weak random source, *Proc. 32nd IEEE Symp. on the Foundations of Comput. Sci.*, pp.79-89, 1991.

†L.Adleman and M.Huang, Recognizing primes in random polynomial time, *Proc. 19th ACM Symp. on Theory of Computing*, pp.462-470, 1987.

$\#reject_{M'}(x)/\#total_{M'}(x) = 1$ である．よって，$\epsilon = \frac{1}{4}$ とおけば，M' が x を受理する必要十分条件は，$\#accept_{M'}(x)/\#total_{M'}(x) > \frac{1}{2} + \epsilon$ かつ M が x を受理することである．ゆえに，$L \in \mathbf{BPP}$.

$\mathbf{RP} \subseteq \mathbf{NP}$ であることは，M を単に NTM と考えればよい．x が RP-TM M に受理されるなら，$\#accept_M(x)/\#total_M(x) > \frac{1}{2}$，すなわち x に対する受理計算が存在するから，x は NTM M に受理される．x が RP-TM M に受理されないなら，$\#reject_M(x)/\#total_M(x) = 1$，すなわち x に対する受理計算は存在しないから，x は NTM M にも受理されない． □

系 7.1 RP-TM の定義において，「ある定数 $0 < \epsilon < \frac{1}{2}$ が存在して」という条件を付け加えた上で，2ヶ所の「$\#accept_M(x)/\#total_M(x) > \frac{1}{2}$」を「$\#accept_M(x)/\#total_M(x) > \frac{1}{2} + \epsilon$」に置き換えてもよい．

問 7.4 RP-TM の定義において，$\frac{1}{2}$ を 1 より小さい任意の正定数で置き換えてもよいことを示せ．

問 7.5 co-$\mathbf{RP} \subseteq \mathbf{BPP}$ であることを示せ．

次の定理は定理 7.4 と同様に証明することができる．

定理 7.6 $L \in \mathbf{RP}$ である必要十分条件は，任意の多項式 $p(n)$ に対し，L を受理する $p(n)$ 時間限定の PTM で次の条件を満たすものが存在することである: $x \in L$ に対するエラー確率はたかだか $(\frac{1}{2})^{p(n)}$，かつ $x \notin L$ に対するエラー確率は 0 である．

問 7.6 定理 7.6 を証明せよ．

また，δ-$\mathbf{RP}$ を δ-$\mathbf{BPP}$ と同様に定義することができ，次の命題が成り立つ:

命題 7.3 任意の $0 < \delta \leq \frac{1}{2}$ に対して，δ-$\mathbf{RP} = \mathbf{RP}$ である．

モンテカルロアルゴリズムとは，“yes” の答には誤りがないような確率性アルゴリズムのことであったが，答が “yes” の場合にも “no” の場合にも誤りがないような確率性アルゴリズムのことを**ラスベガスアルゴリズム** (Las Vegas algorithm) という[‡]．このようなアルゴリズムを形式的に定義するために，受理状態・拒否状態以外に，“わからない” ということを表わす状態 (don't-know state) “?” を持った PTM を考える．

PP-TM M が **ZPP-TM** (ZPP は zero probability polynomial time の頭文字) であるとは，

[‡]モンテカルロアルゴリズムとラスベガスアルゴリズムの違いを初めて考察したのは *IEEE Symp. on Foundations of Comput. Sci.*, 1979 における L.Babai の講演であった．

- M の葉の状態は“受理”か“拒否”か“(?) わからない”のどれかである．
- 任意の x に対して，
  「$\#accept_M(x)/\#total_M(x) > \frac{1}{2}$ かつ $\#reject_M(x)\ /\#total_M(x)=0$」
  または
  「$\#reject_M(x)/\#total_M(x)>\frac{1}{2}$ かつ $\#accept_M(x)\ /\#total_M(x)=0$」
  が成り立ち，かつ
- $\#accept_M(x)/\#total_M(x) > \frac{1}{2}$ なら M は x を受理し，
- $\#reject_M(x)/\#total_M(x) > \frac{1}{2}$ なら M は x を拒否する

が成り立つことである．ZPP-TM が受理する言語のクラスを **ZPP** で表わす．

定理 7.7 $\mathbf{ZPP} = \mathbf{RP} \cap co\text{-}\mathbf{RP}$.

証明 ZPP-TM の“?”状態をすべて拒否状態とした RP-TM を考えることにより，$\mathbf{ZPP} \subseteq \mathbf{RP}$ が証明される．**ZPP** は明らかに補集合で閉じているから，$\mathbf{ZPP} \subseteq$ co-**RP** である．

$\mathbf{RP} \cap$ co-$\mathbf{RP} \subseteq \mathbf{ZPP}$ を示すために，$L, L^c \in \mathbf{RP}$ とする．ここに，L^c は L の補集合である．L, L^c を受理する RP-TM をそれぞれ M_1, M_2 とする．次の PTM M を考える．

```
procedure M(x)
begin
    x を入力として M1 を模倣する;
    if M1 が x を受理 then x を受理する end;
    x を入力として M2 を模倣する;
    if M2 が x を受理 then x を拒否する else "?" 状態で停止する end
end
```

M_1, M_2 が RP-TM であることから，$x \in L$ なら $\#accept_M(x)/\#total_M(x) > \frac{1}{2}$ かつ $\#reject_M(x)/\#total_M(x) = 0$ であり，$x \notin L$ ならば $\#accept_M(x)\ /\#total_M(x) = 0$ かつ $\#reject_M(x)/\#total_M(x) > \frac{1}{2}$ である．よって，M は L を受理する ZPP-TM である． □

系 7.2 $\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{NP} \cap co\text{-}\mathbf{NP}$.

この節で述べた確率性計算量のクラスの間の関係を図 7.4 にまとめた．$A \to B$ は $A \subseteq B$ を表わしている．ここに示した包含関係はどの 1 つも $\subsetneq$ であるかどうか分かっていない．

確率性アルゴリズムは，演習問題 2 に記したように，**P** に属する問題に対しても，よりシンプルなアルゴリズムを提供するという意味で有効であるが，一般的には，**NP** には属すが **P** に属すかどうかも **NP** 完全であるかどうかも分

かっていないような問題(素数性判定問題，多項式の等価性判定問題(演習問題3参照)，グラフの同型性判定問題など)を実際的に解くための苦肉の策ともいえる．その中でも，素数性判定問題は **P** に属すであろうと多くの研究者が信じており，すでに述べたように，ZPPアルゴリズムを持つことが分かっているのに対して，2つのグラフが同型であるか否かを判定する問題はBPPアルゴリズムが存在するかどうかも分かっていない(グラフの同型性判定問題が **NP** 完全なら $\Sigma_2^p = \Sigma_3^p$ が成り立つことが分かっている程度である．Σ_n^p については，第5章の演習問題14を参照せよ)．しかし一般的には，これまで見てきたように，BPPアルゴリズムは実用的な意味で十分有効であると言ってもよい．では，それはどの程度有効なのであろうか？ 例えば，**NP** 完全問題を解くBPPアルゴリズムが存在するのであろうか？ すなわち，$\mathbf{NP} \subseteq \mathbf{BPP}$ が成り立つであろうか？ この問題は現在のところ未解決であるが，$\mathbf{NP} \not\subseteq \mathbf{BPP}$ と予想するに足る状況証拠がいくつもある．例えば，

$$\mathbf{NP} \subseteq \mathbf{BPP} \text{ なら } \Sigma_2^p = \Sigma_3^p \text{ である}$$

が成り立つことが知られているが[§]，未解決ではあるものの $\Sigma_n^p \neq \Sigma_{n+1}^p$ であろうと強く予想されている．相対化(第5章の演習問題4参照)の観点からは，$\mathbf{P} \neq \mathbf{ZPP} \neq \mathbf{NP}$ とか $\mathbf{BPP} \not\subseteq \mathbf{NP}$ を予想させる結果が知られている[¶]．**PP** の位置付けについては，演習問題8を参照せよ．

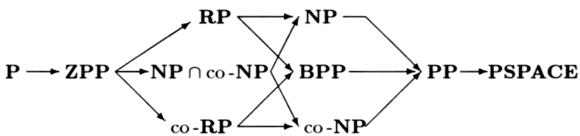


図 7.4　確率性計算量のクラスの間の包含関係

7.2　対話型証明系

数学の教師の悩みの種は，むずかしい定理の証明をいかにして生徒に理解させるかということである．定理の証明を知っているという意味では先生は万能

[§] C.Lautemann, BPP and the polynomial time hierarchy, *Inform. Proc. Lett.* 17, pp.215-218, 1983. この論文では，$\mathrm{BPP} \subseteq \Sigma_2^p \cap \Pi_3^p$ が証明されている．

[¶] L.J.Stockmeyer, On approximation algorithms for #P, *SIAM J. Comput.* 14, pp.849-861, 1985.

J.L.Balcazar and D.A.Russo, Immunity and simplicity in relativizations of probabilistic complexity classes, *Informatique Théorique et Applications* 22, pp.227-244, 1988.

であるのに対して，あまりむずかしいことは理解できない (あるいは，理解しようという努力をしない) ごく普通の能力しか持たない生徒がいたとしよう．そんな生徒に定理が正しいことを理解してもらうには，生徒が納得するまであらゆる質問を受け，それに対し生徒にも分かるような答を返す，という方法が考えられる．(先生をプログラム = TM として) このような方法で "証明" を理解してもらうシステムを対話型証明系という．先生のことを prover (証明する人: 証明を完全に理解している者)，生徒のことを verifier (確認する人: 証明を聞いて納得する者) という．

例 7.2 定理 5.2 で述べたように，**NP** に属す問題 (例えば，与えられたグラフ G がハミルトン閉路を持つかどうかを判定する問題 HAMILTON や，与えられた整数 n が合成数かどうかを判定する問題 COMPOSITE など) は，それを一般的に解くことはむずかしい (すなわち，多項式時間アルゴリズムが知られていない = 証明がむずかしい) が，予想した答 (例えば，HAMILTON では頂点の列 $v_{i_1}, \ldots, v_{i_n}$; COMPOSITE では約数の候補たる 2 つの整数 ℓ, m) が解である ($v_{i_1}, \ldots, v_{i_n}$ が G のハミルトン閉路である; $n = \ell m$ である) かどうかを判定することはやさしい (すなわち，多項式時間で判定できるので生徒にも理解できる)．証明したい事柄 (この例では，グラフ G がハミルトン閉路であること; 整数 n が合成数であること) が与えられたとき，G がハミルトン閉路である場合や n が合成数である場合には，先生が G のハミルトン閉路 $v_{i_1}, \ldots, v_{i_n}$ や 2 つの約数 ℓ, m を生徒に示せば，生徒は多項式時間でそれを確認して納得することができる． □

与えられたグラフがハミルトン閉路を<u>持たない</u>か否かを判定する問題 (NON-HAMILTON と呼ぶ．これは，ハミルトン閉路を<u>持つ</u>か否かを判定する問題である HAMILTON の**補問題** (complement) である) や，与えられた整数が素数かどうかを判定する問題 (PRIME: これは COMPOSITE の補問題である) は，計算量的にそれぞれ HAMILTON, COMPOSITE とは違ったむずかしさを持っていることに注意したい．HAMILTON が **NP** 完全であるのに対して，NON-HAMILTON は co-**NP** 完全であり，NON-HAMILTON $\in$ **NP** かどうかは分かっていない．"グラフの同型性判定問題" など，同類の問題がいくつも知られている (演習問題 10 参照)．後述するように (例 7.3 参照)，対話型証明系はそういった問題を扱えるところに 1 つの意味がある．一方，PRIME とその補問題 COMPOSITE は，ともに **NP** ∩ co-**NP** に属し，ともに **P** に属すのではないかと予想されている．

実は，例 7.2 では先生が一方的に証明を示して生徒に "確認せよ" と言っているだけで，先生と生徒の間できちんとしたコミュニケーションが行なわれてい

ない不完全な“対話”型証明系であるが，まずは，これを定式化することから始めよう[‖].

決定性の対話型証明系 (DIP : deterministic interactive proof system) とは，以下の条件を満たす 2 つの DTM M_P (**証明マシン** (prover) と呼ばれる) と M_V (**検証マシン** (verifier) と呼ばれる) とから成るシステム (M_P, M_V) のことである (図 7.5 参照) :

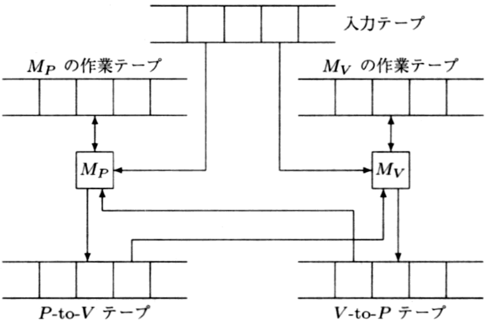


図 7.5 対話型証明系

1. M_P は計算能力に制限のない DTM である.
2. M_V は多項式時間限定の DTM である．ただし，下記の項 5 に記した動作のうち，M_V が動作モードにいる時に使った時間 (M_P の動作モード時の時間は含まない) の総計が M_V の時間量である.
3. M_P と M_V は共通の入力テープを持ち (x を入力としよう)，それぞれ独自の作業用テープを持つ.
4. M_V は出力専用テープ (V-to-P テープ) を持ち，M_V がこのテープに出力した文字列 (M_V から M_P への質問) を M_P は読むことができる．また，M_P も出力専用テープ (P-to-V テープ) を持ち，M_P がこのテープに出力した文字列 (M_V の質問に対する M_P の回答) を M_V は読むことができる.

‖対話型証明系は次の 2 つの論文で独立に導入された．前者は NP に属すかどうか分かっていない一連の代数的問題が属す計算量のクラスを定義するために，後者は暗号系のプロトコル研究のために，この概念を導入した.

L.Babai, Trading group theory for randomness, *Proc. ACM Symp. on Theory of Comput.*, pp.421-429, 1985.

S.Goldwasser, S.Micali and C.Rackoff, The knowledge complexity of interactive proof systems, *SIAM J. Comput.* 18, pp.186-208, 1989. (この論文のオリジナルは 1985 年の ACM Symp. on Theory of Comput. で発表された．)

5. M_P と M_V は交互に"動作可能モード"に入る(ただし，証明系の動作開始時には M_V が先に動作可能モードに入るものとする)．動作可能モードにおいては，入出力テープと作業用テープを使って通常の DTM としての動作を行なう．
6. $p(n)$ を多項式とする．M_V に対しても M_P に対しても，相手の DTM に渡すために出力テープに書き出すことのできる文字列の長さは $p(|x|)$ 以下に制限されている．
7. M_V は自分が動作可能モードにいるとき，受理状態あるいは拒否状態に入って停止することができる．

M_V が入力 x を受理/拒否して停止したとき (M_P, M_V) は x を受理/拒否すると定義する．対 (M_P, M_V) は対話する 2 者の間のコミュニケーションの仕方 (手順) を示しており，**プロトコル** (protocol) と呼ばれる．次の条件を満たす検証マシン M_V が存在するとき，言語 L は **DIP によって証明可能** (L has (admits) a deterministic interactive proof system) であるという:

1. ある証明マシン M_{P^*} が存在して，(M_{P^*}, M_V) は $x \in L$ なるすべての x を受理する．
2. どんな証明マシン M_P を選んでも，(M_P, M_V) は $x \notin L$ なるすべての x を拒否する．

条件 1 は，もし $x \in L$ なら，そのことを M_V が簡単に検証できるような方法が存在する (すなわち，正しい事柄は簡単に証明できる) ことを言っている．条件 2 は，$x \notin L$ なら，"$x \in L$ である" ことを M_V が検証できるような手段はない (すなわち，正しくない事柄は証明できない) ことを言っている．

DIP によって証明可能な言語のクラスを **DIP** で表わす．例 7.2 から次の定理が示唆される．

定理 7.8 DIP = NP.

証明 (**NP** $\subseteq$ **DIP**): $L \in$ **NP** とすると，定理 5.2 より，L を多項式時間で検証する変換機 M が存在する．証明マシンの計算量には制限がないから，$x \in L$ なら x の確認証 y ($|y|$ は $|x|$ の多項式) を P-to-V テープに出力する証明マシン M_{P^*} を M から構成することができる．P-to-V テープから y を読み取って $x \in L$ かどうかを判定する ($x \in L$ のときだけ x を受理する) 多項式 ($|x|$ の多項式) 時間限定の DTM M_V を M から構成することができる．$x \notin L$ なら，M_V の定義より，どんな証明マシン M_P を使っても，M_V が $x \in L$ を受理することはない．

(**DIP** ⊆ **NP**): $L \in$ **DIP** とし，M_V を L に対する検証マシンとする．入力 x に対し，M_P と M_V の間でやり取りする文字列は長さが $|x|$ の多項式であり，かつ個数も $|x|$ の多項式である．よって，M_V の動作を模倣することと，証明マシンの回答を非決定的に推測することとを交互に行なうように，多項式 ($|x|$ の多項式) 時間限定の NTM N を構成することができる．M_V が x を受理するときだけ N も x を受理するようにすれば，$L(N) = L$ である ($x \in L$ なら少なくとも M_{P^*} からの回答を正しく推測できた場合に対応する M_V の計算が x を受理するし，$x \notin L$ なら M_V は決して x を受理しない)． □

検証マシン M_V として DTM の代わりに確率性 TM (BPP-TM) を使うことにより DIP (M_P, M_V) を拡張し，**対話型証明系** (IP : interactive proof system) と呼ぶ:

- $\#accept_{M_V}(x)/\#total_{M_V}(x) > \frac{2}{3}$ なら (M_P, M_V) は x を受理する．
- $\#accept_{M_V}(x)/\#total_{M_V}(x) \le \frac{1}{3}$ なら (M_P, M_V) は x を拒否する．

受理の基準として $\frac{2}{3}$ を採用することに特別の意味はない．$\frac{1}{2}$ より大きい値なら何でもよい．IP によって証明される言語のクラスを **IP** で表わす．

問 7.7 受理基準が $\frac{1}{2}$ より大きいいかなる値であってもクラス **IP** は変わらないことを証明せよ．

例 7.3 与えられた 2 つのグラフ G_0 と G_1 が同型か否かを決定する問題を**グラフの同型性判定問題** (**GRAPH-ISO** : graph isomorphism problem) という．この問題は **NP** に属すが (問: 証明せよ)，その補問題 (**GRAPH-NONISO**: 与えられた 2 つのグラフが同型でないか否かを判定する問題) は **NP** に属すかどうか現在のところ分かっていない．GRAPH-NONISO ∈ **IP** であることを示そう．

図 7.6 の検証マシン M_V を考える[†]．M_V は多項式時間限定である (H を生成するには，G_i の頂点をランダムに置換すればよい)．

まず，任意の同型でないグラフの対 (G_0, G_1) に対し IP (M_{P^*}, M_V) が (G_0, G_1) を受理するような，証明マシン M_{P^*} が存在することを示す:

> **procedure** $M_{P^*}(G_0, G_1)$
> **begin**
> 　検証マシンから H を受け取る;
> 　**if** H と G_0 が同型 **then** 検証マシンに 0 を渡す **else** 1 を渡す **end**
> **end**

[†]O.Goldreich, S.Micali and A.Wigderson, Proofs that yields nothing but their validity and a methodology for protocol design, *Proc. IEEE Symp. on Foundations of Comput. Sci.*, pp.174-187, 1986.

```
procedure M_V(G_0, G_1)
begin
    success ← true;
    for n ← 1 to 2 do /* 2回繰り返す */
        i ← random(0,1); /* i を 0 か 1 にランダムに設定する */
        G_i と同型なグラフ H をランダムに作る;
        H を証明マシンに渡す;
        /* 証明マシンが回答をよこすまで M_V は休止する */
        証明マシンから回答の j を受け取る;
        if i ≠ j then success ← false end
    end;
    if success then 入力を受理する else 拒否する end
end
```

図 7.6 GRAPH-NONISO に対する検証マシン M_V

もし G_0 と G_1 が同型でないなら，M_{P^*} は必ず正しい値を返す (証明マシンの計算能力に制限はないので，そのようにできる) ので，検証マシンは確率 1 で (G_0, G_1) を受理する．

逆に，G_0 と G_1 が同型の場合，証明マシンは H が G_0 から作られた $(i = 0)$ のか G_1 から作られた $(i = 1)$ のか知る由もないので，証明マシンの返す回答 j が i と等しい確率は $\frac{1}{2}$ である．検証マシンと証明マシンの間ではメッセージの交換を 2 回行なうので，検証マシンが (G_0, G_1) を受理する確率はたかだか $\frac{1}{4}$ である． □

GRAPH-NONISO $\notin$ **NP** であろうと予想されているので，この例から **NP** $\subseteq$ **IP** であろうと予想されるが，事実，それは正しい‡:

命題 7.4 **IP** = **PSPACE**.

7.3 近似アルゴリズム

7.3.1 ε 近似

グラフ $G = (V, E)$ の**頂点被覆** (vertex cover) とは，$uv \in E$ なら $u \in C$ または $v \in C$ であるような，V の部分集合 C のことである．5.2 節で述べたように，与えられたグラフ G の頂点被覆で頂点数が最小のものを求める問題 (VERTEX-COVER) は **NP** 完全である．しかし，最小ということにこだわら

‡A.Shamir, IP=PSPACE, *Proc. 31st IEEE Symp. on Foundation of Comput. Sci.*, pp.11-15, 1990.

なければ，次のような多項式時間アルゴリズムが存在する[§].

```
procedure approximate_vertex_cover(G)
begin
    /* G = (V, E) とする */
    C ← 0;
    while E ≠ ∅ do
        uv ∈ E なる任意の辺 uv を選ぶ; C ← C ∪ {u, v};
        /* G から頂点 u, v とそれに接続する辺とを取り除く */
            V ← V − {u, v};
            E ← E − {uw, vw | w ∈ V};
    end;
    C を出力する
end
```

補題 7.3 approximate_vertex_cover(G) は G の頂点被覆を出力する多項式時間アルゴリズムで，出力された C は $|C| \leq 2|G$ の最小頂点被覆$|$ を満たす.

証明 approximate_vertex_cover(G) が出力する C が G の (最小とは限らない) 頂点被覆であることは明らかである．また，このアルゴリズムの実行時間は $O(|E|) = O(|V|^2)$ である.

C は G の辺となる頂点の対をちょうど $|C|/2$ 個含み，それらのどの 2 つも頂点を共有していないことに注意する．G のどんな頂点被覆も，これらの各辺の端点である頂点を少なくとも 1 つ含んでいなければならない (そうでないと，その辺は被覆されていないことになってしまう) から，$|G$ の最小頂点被覆$| \geq |C|/2$ である. □

一般に，複数個存在する解 (**可能解** (feasible solution) という) の中から，ある基準のもとで最大あるいは最小な解 (**最適解** (optimal solution) という) を求める問題を**最適化問題** (optimization problem) という.

P を最適化問題とする．P のインスタンス x に対し，$\mathcal{F}(x)$ を x の可能解の集合とする．cost : $\mathcal{F}(x) \to \mathbf{R}_+$ とする ($\mathbf{R}_+$ は正の実数の集合)．$s \in \mathcal{F}(x)$ に対し，cost(s) を s の**コスト** (cost) という．最小化問題の場合，

$$\mathrm{OPT}(x) = \min\{\mathrm{cost}(s) \mid s \in \mathcal{F}(x)\}$$
$$(\text{最大化問題の場合には，} \mathrm{OPT}(x) = \max\{\mathrm{cost}(s) \mid s \in \mathcal{F}(x)\})$$

を x の最適コストといい，cost(opt) = OPT(x) を満たす解 opt を x の**最適解**という．P を解くアルゴリズムの 1 つを M とすると，M は可能解の 1 つ

[§] D.S.Johnson, Approximation algorithms for combinatorial problems, *J. Comput. System Sci.* 9, pp.256-278, 1974.　この論文は NP 完全問題の近似を系統的に研究した最初の論文である.

$M(x) \in \mathcal{F}(x)$ を出力する (これは，最適解に対する 1 つの**近似解** (approximate solution) である)．$\varepsilon \geq 0$ を定数とする．任意の x に対して

$$\frac{|\mathrm{cost}(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), \mathrm{cost}(M(x))\}} \leq \varepsilon$$

が成り立つとき，M は問題 P の ε **近似アルゴリズム** (ε-aproximation algorithm) であるという．直観的には，この式は近似解 $M(x)$ の最適解に対する相対的な誤差を表わしたものである．分母を $\mathrm{OPT}(x)$ ではなく $\max\{\mathrm{OPT}(x), \mathrm{cost}(M(x))\}$ としているのは，最小化問題と最大化問題とが対称的になるように定義するためである．すなわち，このような定義のもとでは，最小化問題でも最大化問題でも ε は 0 と 1 の間の値をとり，最小化問題の場合には近似解は最小解の $1/(1-\varepsilon)$ 倍以下であり，最大化問題の場合には近似解は最大解の $1-\varepsilon$ 倍以下である (いずれの場合も，ε が 0 に近いほど，良い近似である)．また，

$$r(n) = \max_{|x|=n}\{\frac{\mathrm{cost}(M(x))}{\mathrm{OPT}(x)}, \frac{\mathrm{OPT}(x)}{\mathrm{cost}(x)}\}$$

を P に対する M の**近似比** (approximation ratio) という．

近似アルゴリズムを考える目的は，効率的なアルゴリズムが知られていない問題の近似解を求める効率的なアルゴリズムを開発することであるから，近似アルゴリズムとしては多項式時間アルゴリズムだけを考える．問題 P の**近似閾値** (approximation threshold) とは，

$$\{\varepsilon \mid \text{P に対する多項式時間の } \varepsilon \text{ 近似アルゴリズムが存在する}\}$$

の上限 (上界の最小値) のことである．補題 7.3 より，次の定理を得る:

定理 7.9 VERTEX-COVER の近似閾値はたかだか $\frac{1}{2}$ である．

一口に **NP** 完全問題とは言っても，それぞれの **NP** 完全問題がさまざまに異なる本質を秘めていることは 5.2 節や 7.1, 7.2 節ですでに見た通りであるが，近似アルゴリズムの観点からも差があることを以下で明らかにする．

節 (= リテラルの論理和: 5.2 節参照) の集合 $\mathcal{C} = \{\phi_1, \ldots, \phi_m\}$ と各節の重み $w(\phi_i) \in \mathbf{N}$ が与えられたとき ($\phi_1, \ldots, \phi_m$ の中には同じものがあってもよいものとする)，$\sum_{i=1}^{m} w(\sigma(\phi_i))$ を最大にする付値 σ (ブール変数への真理値 0/1 の割り当て) を求める問題を **MAXSAT** と呼ぶ．ここに，$\sigma(\phi_i)$ は付値 σ のもとでの ϕ_i の値 (0 または 1) を表わす．換言すると，MAXSAT とは，節の有限集合 $\mathcal{C}$ と正整数 k が与えられたとき，$\sum_{i=1}^{m} w(\sigma(\phi_i)) \geq k$ を満たす付値 σ が存在するか否かを決定する問題である．とくに，「すべての節の重みが 1」という特別の場合は，できるだけ多くの節を充足させるような付値を求める問題である．さらに，すべての節がたかだか k 個のリテラルしか含まない場合を **MAX*k*SAT** という．MAX2SAT は **NP** 完全である．

問 7.8 MAX2SAT が **NP** 完全であることを証明せよ．ヒント: 次の 10 個の節を考える: x, y, z, w, $\neg x \vee \neg y$, $\neg y \vee \neg z$, $\neg z \vee \neg x$, $x \vee \neg w$, $y \vee \neg w$, $z \vee \neg w$. これらは次のような性質を満たす．$x \vee y \vee z$ を充足させるような任意の付値は，w の値を適当に選ぶとこの 10 個の節のうちの 7 個を充足するように拡張できるが，w の値をどのように選んでも 8 個を充足させることはできない．また，$x \vee y \vee z$ を充足しないような付値は，この 10 個のうちの 6 個を充足するようにしか拡張できない．この性質を使って，3SAT を MAX2SAT へ還元する．3-CNF $\phi = \phi_1 \wedge \cdots \wedge \phi_m$ に対し，$\phi_i = (\alpha \vee \beta \vee \gamma)$ なら上記の 10 個の節を $\mathcal{C}_\phi$ の元とする，として集合 $\mathcal{C}_\phi$ を定義する．ただし，x, y, z にはそれぞれ α, β, γ を代入し，w には i ごとに異なる新しいブール変数 w_i を代入する．$k = 7m$ とせよ．

MAXSAT をさらに一般化した問題を考えよう．k を正整数とする．節とは限らない一般のブール式 (ただし，ブール変数をたかだか k 個しか含んでいないもの) を元とする集合を $\mathcal{C}$ とする (実は，ブール変数をちょうど k 個含んでいるようなブール式だけに限定してもよい)．$\mathcal{C}$ の元であるブール式をできるだけ多く充足させるような付値を求める問題を **k-MAXGSAT** と呼ぶ (G は generalization の頭文字)．次の定理に述べるように，一般の MAXGSAT に対しては近似閾値が小さい近似アルゴリズムは知られていないが，MAXGSAT の特別の場合である MAXSAT に対しては近似閾値が $\frac{1}{2}$ の近似アルゴリズムが存在し，さらに特別な場合には近似閾値が 2^{-k} の近似アルゴリズムも存在する．

定理 7.10 k-MAXGSAT の近似閾値はたかだか $1 - 2^{-k}$ である．とくに，MAXSAT の近似閾値はたかだか $\frac{1}{2}$ である．もし，すべての節が k 個以上の異なるリテラルを含んでいるような場合には，MAXSAT の近似閾値はたかだか 2^{-k} である．

証明 (1) $\mathcal{C} = \{\phi_1, \ldots, \phi_m\}$ をブール式の集合とする．ϕ_i が k_i 個のブール変数を含むとき，ϕ_i に対する付値は 2^{k_i} 通りあるが，それらのうち ϕ_i を充足させるものの個数 $t(\phi_i)$ は容易に ($O(1)$ 時間で) 計算できる．$p(\phi_i) = t(\phi_i)/2^{k_i}$, $p(\mathcal{C}) = \sum_{i=1}^{m} p(\phi_i)$ とおくと，$p(\phi_i)$ はランダムに選んだ付値が ϕ_i を充足する確率を，$p(\mathcal{C})$ は充足されるブール式の個数の期待値を表わしている．

x をブール変数，ϕ をブール式，$b \in \{0, 1\}$ とするとき，ϕ の中に現れるすべての x を b で置き換えて得られるブール式を $\phi[x = b]$ で表わす．$\mathcal{C}$ がブール式の集合のとき，$\mathcal{C}[x = b] = \{\phi[x = b] \mid \phi \in \mathcal{C}\}$ と定義する．図 7.7 のアルゴリズムを考える．

明らかに，approximate_MAXGSAT は多項式時間限定である．$\phi_1, \ldots, \phi_m$ すべてが充足不能の場合にはいかなる付値も (したがって，approximate_MAXGSAT($\mathcal{C}$) が求めた付値も) 最適解となるので問題ない．そこで，以下ではそれ以外の場合，すなわち $\phi_1, \ldots, \phi_m$ の中に少なくとも 1 つ充足可能なものがある場合を考える．

```
procedure approximate_MAXGSAT(C)
begin
    /* C はブール変数 x_1, ..., x_n を含むとする */
    1. C_1 ← C;
    2. for i = 1 to n do
        2.1. p_0 ← p(C_i[x_i = 0]); p_1 ← p(C_i[x_i = 1]);
        2.2. if p_0 ≥ p_1 then b_i ← 0 else b_i ← 1 end;
        2.3. C_{i+1} ← C_i[x_i = b_i]
    3. end;
    4. σ : (x_1, ..., x_n) ↦ (b_1, ..., b_n) を付値とする
end
```

図 **7.7** k-MAXGSAT を近似するアルゴリズム

for 文実行中の任意の時点における $\mathcal{C}_i$ に対し,

$$p(\mathcal{C}_{i+1}) = \tfrac{1}{2}(p(\mathcal{C}_i[x_i = 0]) + p(\mathcal{C}_i[x_i = 1]))$$

が成り立つ (問: なぜか?). したがって, 行 2.1~2.2 における b_i の決め方より, $p(\mathcal{C}_{i+1}) \geq p(\mathcal{C}_i)$ が成り立つ. とくに, for 文の実行終了時には $p(\mathcal{C}_{n+1}) \geq p(\mathcal{C})$ である. 一方, $p(\mathcal{C}_{n+1}) = \sum_{i=1}^{m} \sigma(\phi_i)/2^0 =$ (最終的に決まった付値 σ のもとで充足される ϕ_i の個数) であり, どの ϕ_i もたかだか k 個しかブール変数を含んでいないのだから $p(\mathcal{C}) \geq \sum_{i=1}^{m} t(\phi_i)/2^k$ である. ゆえに,

$$\begin{aligned}&\text{cost(approximate_MAXGSAT}(\mathcal{C})\ \text{によって得られた解})/\text{OPT}(\mathcal{C})\\ =\ &\textstyle\sum_{i=1}^{m} \sigma(\phi_i)/\sum_{i=1}^{m} t(\phi_i)\ \geq\ 2^{-k}\end{aligned}$$

である. よって, approximate_MAXGSAT の近似閾値はたかだか $1 - 2^{-k}$ である.

MAXSAT の場合にはすべての ϕ_i が節なので $p(\mathcal{C}) = \sum_{i=1}^{m} p(\phi_i) \geq \frac{1}{2}$ であるから, 近似閾値はたかだか $1 - \frac{1}{2} = \frac{1}{2}$ である. また, MAXSAT のすべての節が k 個以上のリテラルを含んでいる場合には, すべてのリテラルを 0 とする付値以外はその節を充足させるから, $p(\mathcal{C}) = \sum_{i=1}^{m} p(\phi_i) \geq 1 - 2^{-k}$ である. よって, 近似閾値はたかだか 2^{-k} である. □

次に, いくらでも良い近似ができる **NP** 完全問題の例を示す.

ナップザック問題 (KNAPSACK) とは, 大きさがそれぞれ $w_1, \cdots, w_n$ で, 価値がそれぞれ $v_1, \cdots, v_n$ である n 個の品物 (それぞれ 1 つずつ) と, 容量が W のナップザックが 1 つ与えられたとき, 制限容量を満たしながら ($\sum_{i \in S} w_i \leq W$), 価値 $\sum_{i \in S} v_i$ が最大になるように, ナップザックに詰める品物を選ぶ ($S \subseteq \{1, 2, \ldots, n\}$) という問題であった (5.2 節).

定理 7.11 KNAPSACK の近似閾値は 0 である. すなわち, 任意の $\varepsilon > 0$ に対し, KNAPSACK の ε 近似アルゴリズムが存在する.

証明 $V = \max\{v_1, \ldots, v_n\}$ とする．n 個の品物のうちの最初の i 個の中から何個か選び，それらの価値の和がちょうど v となるようにしたい．そのような制限のもとでの重さの和の最小値を $W(i,v)$ で表わす．すなわち，

$$W(i,v) = \min\{\textstyle\sum_{i\in S} w_i \mid S \subseteq \{1,\ldots,i\},\ \sum_{i\in S} v_i = v\}.$$

次の関係が成り立つ：

$$\begin{cases} W(0,v) = \infty \\ W(i+1,v) = \min\{W(i,v),\ W(i, v-v_{i+1}) + w_{i+1}\}. \end{cases}$$

$W(n,v) \leq W$ であるような最大の $v \leq nV$ を与える S が求める解である．これは $O(n^2V)$ 時間で求めることができるが，残念ながら n^2V は入力のサイズの多項式ではない (入力のサイズは $s = |w_1| + \cdots + |w_n| + |W| + |v_1| + \cdots + |v_n|$ とするのが妥当であろう．ここに，$|x|$ は整数 x の桁数である)．しかし，われわれが欲しいのは厳密解ではなく近似解であるから，v_i $(i = 1, \ldots, n)$ の代わりに，v_i の 2 進数表現の下位 b ビットを切り捨てた値 $v'_i = \lfloor v_i/2^b \rfloor 2^b$ を用いれば，実行時間は $O(n^2V/2^b)$ になる．本来の問題のインスタンス $x = (w_1, \ldots, w_n, W, v_1, \ldots, v_n)$ に対する最適解を $S \subseteq \{1, \ldots, n\}$ とし，修正したインスタンス $x' = (w_1, \ldots, w_n, W, v'_1, \ldots, v'_n)$ に対する解を S' とすると，一連の不等式

$$\sum_{i\in S} v_i \geq \sum_{i\in S'} v_i \geq \sum_{i\in S'} v'_i \geq \sum_{i\in S} v'_i \geq \sum_{i\in S} (v_i - 2^b) \geq \sum_{i\in S} v_i - n2^b$$

が成り立つ．最初の不等号は S が x の最適解であることによる．2 番目の不等号は $v_i \geq v'_i$ より，3 番目の不等号は S' が x' の最適解であることより，4 番目の不等号は $v'_i \geq v_i - 2^b$ より，最後の不等号は $|S| \leq n$ より，それぞれ導かれる．この不等式より，(最適解のコスト − 近似解のコスト) $\leq n2^b$ が得られる．一方，厳密最適解の下界は V である (任意の i に対して $w_i \leq W$ であると仮定しても，ナップザック問題の一般性は失われない) から，

$$\frac{\text{最適解のコスト} - \text{近似解のコスト}}{\text{最適解のコスト}} \leq n2^b/V = \varepsilon$$

が導かれる．

よって，$\varepsilon > 0$ が任意に与えられたとき，$b = \lceil \log_2 \frac{\varepsilon V}{n} \rceil$ とすれば，上述の方法は KNAPSACK に対する $O(n^2V/2^b) = O(n^3/\varepsilon)$ 時間限定の ε 近似アルゴリズムになっている． □

最後に，近似アルゴリズムの限界を示す例として，いかなる ε 近似アルゴリズムも持たない **NP** 完全問題が存在することを示す．

定理 7.12 $\mathbf{P} \neq \mathbf{NP}$ なら，TSP (5.2 節参照) の近似閾値は 1 である．

証明 TSP に対する多項式時間 ε 近似アルゴリズム $(0 < \varepsilon < 1)$ が存在したとすると，ハミルトン閉路問題 HAMILTON が多項式時間で解けてしまうことを以下で示す．HAMILTON は **NP** 完全問題である (定理 5.6) から，これは $\mathbf{P} = \mathbf{NP}$ を意味し，証明が完了する．

グラフ $G = (V, E)$ が与えられたら，まず，次のような TSP のインスタンス x_G を構成する．x_G は $|V|$ 個の頂点を持つ重み付き完全グラフで，w をその重み関数とするとき，辺 uv の重みは

$$w(uv) = \begin{cases} 1 & uv \in E \text{ のとき} \\ |V|/(1-\varepsilon) & uv \notin E \text{ のとき} \end{cases}$$

と定義されているものとする．x_G が構成できたら，TSP に対する ε 近似アルゴリズムをこれに適用する．もし，この ε 近似アルゴリズムによって得られたハミルトン閉路の重みの和が $|V|$ に等しかったら，このハミルトン閉路は G のハミルトン閉路でもある．もしそうでなかったら (すなわち，この ε 近似アルゴリズムによって得られたハミルトン閉路上の少なくとも 1 つの辺の重みが $|V|/(1-\varepsilon)$ だったら)，この閉路の重みの和 W は，$W > |V|/(1-\varepsilon)$ を満たす．一方，ε 近似の定義より

$$\frac{W - \mathrm{OPT}(x_G)}{W} \leq \varepsilon$$

であるから，$\mathrm{OPT}(x_G) \geq (1-\varepsilon)W$ である．よって，$\mathrm{OPT}(x_G) > |V|$ である．これは G にハミルトン閉路が存在しないことを意味する (もし存在したとすると，それは x_G のハミルトン閉路でもあり，しかもその重みの和が $|V|$ であるから，$\mathrm{OPT}(x_G) > |V|$ に矛盾する)．

このように，G がハミルトン閉路を持つかどうかを，x_G の多項式時間近似アルゴリズムを使って判定することができる． □

問 7.9 すべての辺の重みが 1 または 2 であるような無向グラフに対する TSP を考える．次の 2 つことが成り立つことが知られている:

(a) TSP をこのように制限しても **NP** 完全のままである¶．

(b) 重み付き完全グラフ G が与えられたとき，次のような G の部分グラフ H を求める多項式時間アルゴリズムが存在する‖: H のどの頂点の次数も 2 であり (したがって，H は独立なサイクルの集合である)，そのような部分グラフの中で H は重みの和が最小のものである．

¶R.M.Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations* (R.E.Miller and J.W.Thatcher ed.), Plenum, 1972.

‖C.H.Papadimitoriou and K.Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982. この本の 17 章は近似アルゴリズムに詳しい．

(1) H に対する TSP の近似アルゴリズムはどれも $\frac{1}{2}$ 近似アルゴリズムであることを示せ．

(2) 上述のように制限された無向グラフにおける TSP に対する $\frac{1}{4}$ 近似アルゴリズムが存在することを証明せよ**．ヒント: H から出発し，2 つのサイクルを結合して次第に大きくしてゆくことを考えよ．重みが 2 の辺がどのサイクル上にもない間は 2 つのサイクルを結合するとその結果重みの和が 2 増えることもあるが，一旦重みが 2 の辺がサイクル上に生じたら，その辺を除去して結合するようにすれば重みの和はたかだか 1 しか増えない．

7.3.2 近似不可能性

A を最適化問題とする．任意の $\varepsilon > 0$ と A の任意のインスタンス x に対して，相対誤差がたかだか ε であるような，x の解を $|x|$ の多項式時間で計算するアルゴリズムが存在するとき，このアルゴリズムのことを A の**多項式時間近似スキーム** (polynomial-time approximation scheme) という．さらに，この多項式が $\frac{1}{\varepsilon}$ に関しても多項式であるとき，この近似スキームは**完全多項式** (fully polynomial) **近似スキーム**であるという．多項式時間近似スキームを持つような最適化問題は，近似の観点からは非常に望ましい性質を持つ問題であるといえる．定理 7.11 の証明より，KNAPSACK は完全多項式近似スキームを持つことがわかる．$\mathbf{P} = \mathbf{NP}$ でない限り，強 **NP** 完全問題 (問題 5.11 参照) は完全多項式近似スキームを持たないことが知られている．例えば，BIN-PACKING は多項式時間近似スキームを持つが，その多項式は $\frac{1}{\varepsilon}$の指数関数である．

この例や前節の結果が示しているように，**NP** 完全問題同士といえども近似可能性の度合が異なる．そのことは，単に対数領域還元性だけでは近似可能性に関して問題を分類することができないことを意味している．そのため，近似可能性を保存するような還元を新たに考える必要がある．

A, B を最適化問題とする．A から B への **L 還元** (L-reduction) とは，次の条件を満たす関数の対 (R, S) のことをいう．

1. R も S も対数領域で計算できる．
2. x を A のインスタンスとすると $R(x)$ は B のインスタンスで，
$$\mathrm{OPT}(R(x)) \le \alpha \cdot \mathrm{OPT}(x)$$
を満たす定数 $\alpha > 0$ が存在する．
3. s が $R(x)$ の可能解ならば $S(s)$ は x の可能解であり，
$$|\mathrm{OPT}(x) - \mathrm{cost}(S(s))| \le \beta \cdot |\mathrm{OPT}(R(x)) - \mathrm{cost}(s)|$$
を満たす定数 $\beta > 0$ が存在する．

**C.H.Papadimitoriou and M.Yannakakis, The traveling salesman problem with distances one and two, *Math. of Operations Research* 18, pp.1-11, 1993.　この論文で $\frac{1}{7}$ 近似アルゴリズムが与えられた．また，この制約された TSP でさえ MAXSNP 困難であることが示された (実は，MAXSNP 完全である)．

すなわち，これらの条件が要求していることは，A の問題 x を解く代わりに B の問題 $R(x)$ を解いて，その解 s を S で変換した $S(s)$ を考えれば，それは x の最適解からそれほどかけ離れていない，ということである．3 番目の条件から，s が $R(x)$ の最適解ならば $S(s)$ は x の最適解である．

例 7.4 独立集合問題 (INDEPENDENT-SET) とは 与えられたグラフ $G = (V, E)$ の独立集合 (どの 2 点 $u, v \in U$ に対しても $uv \notin E$ であるような，V の部分集合 U のこと) で頂点数が最大のものを求める問題である (5.2 節参照)．INDEPENDENT-SET から VERTEX-COVER への自明な還元 (R は恒等関数，S は C を $V - C$ に変換する関数) は L 還元ではない．なぜなら，最適な頂点被覆が最適な独立集合よりいくらでも大きくなりうる (例えば，大きなクリークを考えよ) からである．

しかし，最大次数が k 以下であるようなグラフだけに限定すると，(R, S) は k-INDPENDENT-SET から k-VERTEX-COVER への L 還元である． □

問 7.10 (1) 例 7.4 の (R, S) は k-INDPENDENT-SET から k-VERTEX-COVER への L 還元であること，また，(R, S) は k-VERTEX-COVER から k-INDPENDENT-SET への L 還元でもあることを示せ．

(2) L 還元が推移律を満たすこと，すなわち，(R, S) が A から B への L 還元で，(R', S') が B から C への L 還元ならば，$(R' \circ R,\ S' \circ S)$ は A から C への L 還元であることを示せ．

近似問題の間での還元性が定義できたので，近似問題に関して **NP** に相当するクラスを定義したい．

$S, R_1, \ldots, R_m$ を有限集合 V 上の述語，$x_1, \ldots, x_k$ を V 上の変数，$\varphi(x_1, \ldots, x_k, S, R_1, \ldots, R_m)$ を $x_1, \ldots, x_k, S, R_1, \ldots, R_m$ に関する論理式 (全称記号 $\forall$ や存在記号 $\exists$ を含まないもの) とするとき，

$$\max_S |\{(x_1, \ldots, x_k) \in V^k \mid \varphi(x_1, \ldots, x_k, S, R_1, \ldots, R_m)\}|$$

の形の式で表現できる最適化問題を $\mathbf{MAXSNP_0}$ **問題**と呼ぶ (SNP は strict **NP** の意)．演習問題 14 参照．

$\mathrm{MAXSNP_0}$ 問題に L 還元可能な最適化問題のクラスを **MAXSNP** で表わす．

例 7.5 (1) グラフ $G = (V, E)$ を 2 つの頂点集合 $(S, V - S)$ に分割することを**カット** (cut) といい，$|\{uv \in E \mid u \in S, v \in V - S\}|$ をこのカットのサイズという．サイズが最小のカットを見つける問題は **P** に属すことが知られている[‡]が，サイズが最大のカットを見つける問題 (**MAX-CUT**) は **NP** 完全である．MAX-CUT は

[‡]例えば，巻末の「参考書案内」の Papadimitoriou-Steiglitz あるいは Cormen-Leiserson-Rivest を見よ．

$$\max_S |\{(x,y) \mid (R(x,y) \vee R(y,x)) \wedge S(x) \wedge \neg S(y)\}|$$

と表わすことができるので，MAX-CUT は MAXSNP_0 に属す．したがって，MAX-CUT は **MAXSNP** にも属す．ここに，$R(x,y)$ は，G を有向グラフと考えて，頂点 x から y への辺があることを表わす述語であり，$S(x)$ は x が S に属すことを表わす述語である．

(2) MAX2SAT が **MAXSNP** に属すことを示そう．V を論理変数の集合とする．述語 R_1, R_2, R_3 を

$$\begin{aligned} R_1(x,y) &\iff x \vee y \text{ が節である}, \\ R_2(x,y) &\iff \neg x \vee y \text{ が節である}, \\ R_1(x,y) &\iff \neg x \vee \neg y \text{ が節である} \end{aligned}$$

と定義すると，MAX2SAT は

$$\begin{aligned} \max_S |\{(x,y) \mid \quad & (R_1(x,y) \wedge (S(x) \vee S(y)) \\ & \vee (R_2(x,y) \wedge (\neg S(x) \vee S(y)) \\ & \vee (R_3(x,y) \wedge (\neg S(x) \wedge \neg S(y))\}| \end{aligned}$$

と表わすことができる．ここに，$S \subseteq V$ は真理値 1 (**true**) を割り当てるべき論理変数の集合である． □

問 7.11 k-INDEPENDENT-SET, k-VERTEX-COVER がそれぞれ **MAXSNP** に属すことを示せ.

$L_0 \in$ **MAXSNP** であり，かつ **MAXSNP** に属すすべての問題が L_0 に L 還元できるとき，L_0 を **MAXSNP 完全** (**MAXSNP**-complete) であるという.

命題 7.5 次の問題はいずれも **MAXSNP** 完全である[§]:

(a) MAX3SAT　(b) MAX-CUT

(c) 4-INDEPENDENT-SET　(d) 4-VERTEX-COVER

最後に, (**P** $\neq$ **NP** という仮定のもとで) TSP 以外にもまったく近似不可能 (近似閾値が 1) の問題，あるいは思うままの近似は不可能 (多項式時間近似スキームを持たない) 問題が存在することを述べてこの章を閉じることにする[¶].

[§]命題 7.5,7.6 の証明については，巻末に示した Papadimitoriou の本の 13 章を参照せよ．次の論文では APX 完全という概念を導入して，3-MAX-CUT, 3-INDEPENDENT-SET, 3-VERTEX-COVER が APX 完全であることを示している:

P.Alimonti and V.Kann, Hardness of approximation problems on cubic graphs, *Lecture Notes in Comput. Sci.* 1203, pp.288-298, 1997.

[¶]S.Arora, C.Lund, R.Motwani, M.Sudan and M.Szegedy, Proof verification and hardness of approximation problems, *Proc. 33rd IEEE Symp. on Foundation of Comput. Sci.*, pp.14-23, 1992 (同じシンポジウムの予稿集 pp.2-13 も参照のこと).

D.S.Johnson, The tale of the second prover, *J. Algorithms* 13, pp.502-524, 1992.

命題 7.6 (1) $\mathbf{P} \neq \mathbf{NP}$ とすると，次の問題はどれも多項式時間近似スキームを持たない:

(a) MAX3SAT (b) MAX2SAT (c) MAX-CUT

(d) 4-INDEPENDENT-SET (e) VERTEX-COVER

(2) $\mathbf{P} \neq \mathbf{NP}$ とすると，次の問題の近似閾値は1である:

(a) INDEPENDNET-SET (b) クリーク問題(5.2節参照)

本書では定数の近似閾値(あるいは，定数の近似比)を持つような問題だけを考えたが，近似閾値や近似比を $|x|$ の関数に拡張することもできる．例えば，$\mathbf{P} = \mathbf{NP}$ でない限り，最小集合被覆問題(5.2節参照)の近似比は $\Theta(\log n)$ であることが知られている‖.

演 習 問 題 7

1 $a \in \Phi(n)$ なら $a^{\phi(n)} \equiv 1 \pmod n$ であることを証明せよ．

2 $n \times n$ 行列 A, B, C が与えられたとき，$AB = C$ か否かを判定する問題を考える．

(1) この問題を解く $O(n^3)$ 時間アルゴリズムを示せ．実は，$O(n^{2.5})$ アルゴリズムが存在することが知られている．

(2) 図7.8に示した確率性アルゴリズムを考える．このアルゴリズムの実行時間は $O(n^2)$ であること，および，このアルゴリズムが誤った答を出力する確率は1/2以下であることを示せ．ヒント: $AB \neq C$ なら，$A(Bx) = Cx$ なる任意の x に対して，$A(By) \neq Cy$ なる y が存在する．

3 $x_1, \ldots, x_n$ の多項式 $p(x_1, \ldots, x_n)$ が与えられたとき，恒等的に $p(x_1, \ldots, x_n) = 0$ であるか否かを判定する問題(**ZERO-POLY**)は，多項式時間アルゴリズムが存在するかどうか現在のところ知られていない(2つの多項式が恒等的に等しいかどうかを判定することはZERO-POLYの特別の場合であることに注意する)．

(1) $p(x_1, \ldots, x_n)$ が d 次の多項式で恒等的に0ではないとき，$p(a_1, \ldots, a_n) = 0$ を満たす n 個の整数の組 $(a_1, \ldots, a_n)$ はたかだか $nd(2nd+1)^{n-1}$ 個しかないことを証明せよ(ただし，各 a_i は $-nd \leq a_i \leq nd$ であるとする)．ヒント: n に関する帰納法.

(2) ZERO-POLYを解く多項式時間モンテカルロアルゴリズムを示せ.

4 PTMが受理する言語のクラスは帰納的可算言語のクラスに一致することを証明せよ．

‖C.Lund and M.Yannakakis, On the hardness of approximating minimization problems, *Proc. 25th ACM Symp. on the Theory of Comput.*, pp.286-295, 1993.

なお，近似アルゴリズムについては，以下の文献が詳しい:

V.Kann, On the approximability of NP-complete optimization problems, Ph.D thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Sweden, 1992.

V.T.Paschos, A survey of approximately optimal solutions to some covering and packing problems, *ACM Comput. Surveys* 29, pp.171-209, 1997.

```
function matrix_multiplicatio(A, B, C)
begin
    for i = 1 to n do
        /* −1, 1 を要素とする配列をランダムに生成する */
        x[i] ← random(0, 1);
        if x[i] = 0 then x[i] ← −1 end
    end;
    if A(Bx) ≠ Cx then
        return "AB = C ではない"
    else
        return "AB = C である"
    end
end
```

図 7.8 $AB = C$? を判定する確率性アルゴリズム

5 (1) **BPP** および **ZPP** は補集合，和集合，共通部分をとる演算で閉じていること ($L_1, L_2 \in$ **BPP** $\Longrightarrow L_1^c, L_1 \cup L_2, L_1 \cap L_2 \in$ **BPP**)，また，**RP** は $\cup, \cap$ で閉じていることを証明せよ．**RP** が ^{c} で閉じているかどうかは分かっていない．**PP** は $^c, \cup, \cap$ それぞれのもとで閉じていることが知られている．

(2) **PP** が $\leq_{poly}$ で閉じていることは補題 7.1 で示した．**BPP**, **RP**, **ZPP** も $\leq_{poly}$ で閉じていることを示せ．

6 対数領域限定の RP-TM によって受理される言語のクラスを **RL** で表わす．**DL** $\subseteq$ **RL** $\subseteq$ **NL** であることを示せ (包含関係が真であるかどうかは分かっていない)．

有向グラフに対する到達可能性問題を GAP というのに対して，無向グラフに対するそれを **UGAP** という．UGAP $\in$ **RL** であること，および **RL** $\subseteq$ **SC** であることが知られている (**SC** については，6.2.3 節を参照のこと) **．

7 $p(n)$ を任意の多項式とする．**確率性 ATM** とは次のような，$p(n)$ 時間限定 ATM M のことをいう．M の状態集合は "+状態" と "MAX 状態" の 2 種類があり (+状態は ATM の存在状態にあたり，MAX 状態は全称状態にあたる)，計算の過程において+状態と MAX 状態はつねに交互に現れる．長さ n の入力 x に対する M の計算木は高さ $2p(n)$ の完全 2 分木であり，葉の状態は受理状態か拒否状態のどちらかである．

各ノードの "受理カウント" を次のように定義する．受理状態の葉の受理カウントは 1，拒否状態の葉の受理カウントは 0 とする．+状態の内部ノードの受理カウントは 2 つの子の受理カウントの和とし，MAX 状態の内部ノードの受理カウントは 2 つの子の受理カウントの大きい方とする．計算木の根における受理カウントが $2^{p(n)-1}$ より大きいとき M は x を受理する．

確率性 ATM によって受理される言語のクラスを **APP** で表わす．**APP** = **PSPACE** であることが知られている‡．

SSAT(stochastic SAT) とは次のような決定問題である: ブール式 $\phi(x_1, \ldots, x_n)$ が与えられたとき，"x_1 をうまく選び"，"x_2 をランダムに選び"，"x_3 をうまく選び"，"x_4 をランダムに選び"，"x_5 をうまく選び"，$\cdots$ として $\phi(x_1, \ldots, x_n)$ の値が 1 になる確率を 1/2 より大きくなるようにできるか?

**N.Nisan, RL⊆SC, *Proc. 24th ACM Symp. on Theory of Comput.*, pp.619-623, 1992.

‡C.H.Papadimitoriou, Games against nature, *J. Comput. System Sci.* 31, pp.288-301, 1985.

SSAT $\in$ **APP** であることを示せ．実は，SSAT は **PSPACE** 完全な問題である．

8 (1) この問題に関しては，5.5 節および第 5 章の演習問題 4, 14, 18 を参照せよ．

#P, $\oplus$**P** と確率性計算量のクラスが深い関係にあることは容易に想像できる．例えば，**#P** に属する問題は NTM の受理計算の個数 (0 と 2^n の間のある値，の n ビットすべて) を求めているのに対して，**PP** に属する問題はその値の先頭の 1 ビットだけを求めていると見ることができる．この意味で，**#P** は **PP** より強力であると言える．

次の関係が成り立つことが知られている[§]:

$$\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{PP}}(\text{戸田の定理}), \quad \mathbf{NP} \subseteq \mathbf{RP}^{\oplus\mathbf{P}}, \quad \mathbf{P}^{\mathbf{PP}} = \mathbf{P}^{\#\mathbf{P}}.$$

このように，確率性 TM は非常に強力な計算能力を持っている．脚注に示した文献を読んでみよ．

(2) $\mathbf{D}^p \subseteq$ **PP** を示せ．$\mathbf{D}^p$ については，5 章の演習問題 15 を見よ．

9 **NC** のランダム版を **RNC**(randomized NC) という．正確には次のように定義する．まず，"確率性論理回路" C とは，通常の入力 $\mathbf{x} = (x_1, \ldots, x_n)$ 以外に "ランダム入力" $\mathbf{z} = (z_1, \ldots, z_m)$ と出力 $\mathbf{y} = (y_1, \ldots, y_\ell)$ を持つ回路である．入力 $\mathbf{x}$ に対する C の出力が $\mathbf{b} \in \{0,1\}^\ell$ に等しい確率 $\mathrm{Prob}[C(\mathbf{x}) = \mathbf{b}]$ を

$\mathrm{Prob}[C(\mathbf{x}) = \mathbf{b}]$

$= |\{(z_1, \ldots, z_m) \mid \text{入力 } (x_1, \ldots, x_n, z_1, \ldots, z_m) \text{ に対する出力が } \mathbf{b} \text{ に等しい}\}| \,/\, 2^m$

で定義する．$\mathrm{Prob}[C(\mathbf{x}) = \varphi(\mathbf{x})] \geq 1 - \varepsilon$ のとき，C は関数 φ を "エラー確率 ε で計算する" という．

任意の整数 $k \geq 1$ に対し，$\mathrm{size}(C_n) = n^{O(1)}$ かつ $\mathrm{depth}(C_n) = O(\log^k n)$ の対数一様な論理回路族 $\mathcal{C}$ がたかだか 1/4 のエラー確率で言語 L の特性関数を計算するとき，$L \in \mathbf{RNC}^k$ であると定義し，$\mathbf{RNC} = \bigcup_{k\geq 1} \mathbf{RNC}^k$ とおく．

(1) **NC** $\subseteq$ **RNC** $\subseteq$ **P** であることを示せ．

(2) 次の各問題が **RNC** に属すことを脚注の文献で調べよ．

(a) 与えられたグラフの極大な独立集合を求める問題[¶]．

(b) 与えられたグラフの，辺数が最大のマッチングを求める問題[‖]

グラフ $G = (V, E)$ の**マッチング** (matching) とは，どの 2 辺も端点を共有しないような $E' \subseteq E$ のことである．

10 互いに素な整数 $n < m$ が与えられたとき，$x^2 \equiv n \pmod m$ を満たす整数 x が存在する否かを決定する問題は **NP** 完全である．この問題の補問題は **IP** に属すことを示せ．ヒント: 検証マシン用のアルゴリズムとして図 7.9 に示したものを考えよ．

11 メルリンはアーサー王お抱えの魔法使いである．アーサーは学問好きな王様として知られ，数学の定理についても，証明を聞いて納得しないと気がすまない性格であった．あるとき，メルリンは 2 つのグラフ G_0, G_1 が同型でないことを発見したとアーサー

[§] S.Toda, On the computational power of PP and ⊕P, *Proc. 30th IEEE Symp. on Foundation of Comput.Sci.*, pp.514-519, 1989.

L.G.Valiant and V.V.Vazirani, NP is as easy as detecting unique solutions, *Theor. Comput. Sci.* 47, pp.85-93, 1986.

[¶] M.Luby, A simple parallel algorithm for the maximal independent set, *SIAM J. Comput.* 15, pp.1036-1053, 1986. 実はその後，この問題は NC^2 に属すことが証明された:

M.Luby, Romoving randomness from parallel computation without processor penalty, *Proc. 29th IEEE Symp. on Found. Comput. Sci.*, pp.162-173, 1988.

[‖] K.Mulmuley, U.V.Vazirani and V.V.Vazirani, Matching is as easy as matrix inversion, *Combinatorica* 7, pp.105-113, 1987.

```
procedure M_V(n, m)
begin
    k ← |m|;
    z_i < b かつ gcd(z_i, b) = を満たす z_1, ..., z_k をランダムに生成する;
    b_i ∈ {0, 1} なる b_1, ..., b_k をランダムに生成する;
    for i ← 1 to k do
        if b_i = 1 then w_i ← z_i^2 (mod m) else w_i ← n z_i^2 (mod m) end
    end;
    w_1, ..., w_k を証明マシンに渡す;
    証明マシンからの回答 c_1, ..., c_k を受け取る;
    for i ← 1 to k do
        if c_i ≠ b_i then 入力を拒否して停止する end
    end;
    入力を受理する
end
```

図 7.9 $x^2 \equiv n \pmod{m}$? 問題のための検証マシン

に報告した．アーサーは多忙なので限られた時間 ((G_0, G_1) のサイズの多項式時間) しか使えない．GRAPH-NONISO は **NP** に属すかどうか分かっていない問題なので，アーサーには多項式時間で G_0 と G_1 が同型でないことを確かめることはできない．しかし，メルリンの報告は嘘であって，メルリンは同型な G_0 と G_1 をアーサーに見せているのかもしれない (GRAPH-ISO は **P** に属すかどうか分かっていないので，アーサーの能力では同型性を判断できない)．

メルリンの報告が嘘でないことを確かめるために，アーサーは次のような方法を思いついた．アーサーはコイン投げをして G_0 か G_1 をランダムに選び，さらに選んだグラフの頂点の順序をランダムに並べ変えたグラフ H を作ってメルリンに見せ，H が G_0 と同型なのかあるいは G_1 と同型なのかを答えよと命じた．魔法使いである超能力者のメルリンには，同型性の判定ができるはずなので，メルリンはどちらであるかを必ず答えなければならない (そうしないと嘘の報告をしたことが直ちにばれてしまう)．報告が嘘でなく G_0 と G_1 が同型でない場合にはメルリンが返してよこす答はつねに正しい．しかし，報告が嘘であって同型である場合には H は G_0 とも G_1 とも同型であるから，メルリンはあてずっぽうでどちらかを答えるしかない．よって，アーサーは G_0 と G_1 が同型でないことを 1/2 の確率で信じることができる．しかし，同じことを 2 回行なえば，アーサーは G_0 と G_1 が同型でないことを 3/4 の確率で信じることができる．

上述したのは GRAPH-NONISO の例であるが，**アーサー・メルリンゲーム** (Arthur-Merlin game) とは，この例のように，検証マシン (アーサー) から証明マシン (メルリン) へ渡せるメッセージとして許されるのは検証マシンがコイン投げを行なって得た結果 ($\{0,1\}^*$ の語) だけであるように，プロトコルがきつく制限された対話型証明系 (ある問題について，メルリンがアーサーを高い確率で納得させることができればメルリンの勝ち，そうでなければアーサーの勝ちとするゲーム) のことである．アーサー・メルリンゲームと IP とは同値であることが知られている[**]．

アーサーとメルリンの間でやり取りするメッセージ交換回数を k 回に限定してメルリンがアーサーを納得させることができる問題 (言語) のクラスを $AM(k)$ で表わす．GRAPH-NONISO $\in AM(4)$ であることを確認せよ (例 7.3)．実は，任意の $k \geq 2$ に対して $AM(k) = AM(2)$ が成り立つ．GRAPH-NONISO $\in AM(2)$ であることを示

[**] S.Goldwasser and M.Sipser, Private coins vs. public coins, in interactive proof systems, *Proc. 18th ACM Symp. on Theory of Comput.*, pp.59-68, 1986.

せ．また，$AM(1)$ はどのような言語のクラスか？

12 n 個の都市と，都市 i, j 間の距離 d_{ij}，および "必ず通らなければならない都市" の集合 $M \subseteq \{1, 2, \ldots, n\}$ が与えられたとき，M を含み重みの和が最小な連結グラフ (当然それは木である) を求める問題を**スタイナー木問題** (Steiner tree problem) という．この問題は **NP** 完全であり，近似閾値は 1 であることが知られている．また，すべての d_{ij} が 1 または 2 であっても **MAXSNP** 完全である．

任意の i, j, k に対して $d_{ik} \le d_{ij} + d_{jk}$ が成り立つ場合，近似閾値はたかだか $\frac{1}{2}$ であることを示せ[††]．ヒント：まず，全頂点間の最短経路を求め，次にそれを使って，必ず通るべき都市のすべてを含むような重み和が最小の木を求める．最後に，それをスタイナー木に拡張する．

13 箱詰め問題 (BIN-PACKING) (5.2 節参照) の近似アルゴリズムとして，最初に適合する箱に順次詰めてゆくという方法を考える．近似比を求めよ．

14 述語 (predicate) とは論理値 (**true**/ **false**) を値とするような関数のことである．述語をベースにして，命題論理を拡張した論理体系を**述語論理** (predicate logic) という．述語論理における論理式では，**true**/**false** の値を取る論理変数や，存在記号 ($\exists$)，全称記号 ($\forall$) 以外に，述語を表わす記号や，関数や定数を表わす記号が使われる．例えば，R が 2 変数の述語を表わす記号，f が 1 変数関数を表わす記号，a が定数を表わす記号，x, y が論理変数であるとき，

$$\forall x \exists y (R(f(x), a) \wedge R(y, f(f(x))))$$

は **1 階** (first-order) 述語論理の論理式である．いま，論理変数が取りうる値は実数だけであるとし，$R(x, y)$ は $x \ge y$ を表わす述語であり，$f(x) = x^2$，$a = 0$ であると "解釈" すると，この論理式は「任意の実数 x に対して実数 y が存在して，$x^2 \ge 0$ かつ $y \ge x^4$ が成り立つ」ことを表わす．ここで，"1 階" の論理式と言っているのは，存在記号や全称記号が論理変数とだけ結び付いていることを意味する．これに対し，存在記号や全称記号に集合を結び付けることも許した論理体系を **2 階** (second-order) の述語論理という．

例えば，グラフ G，$x, y \in V(G)$, $C \subseteq V$ に対して，$E(x, y) = \textbf{true} \overset{\text{def}}{\Longleftrightarrow} xy \in E(G)$, $C(x) = \textbf{true} \overset{\text{def}}{\Longleftrightarrow} x \in C$ とすれば，

$$\exists C \forall u \forall v (E(u, v) \rightarrow C(u) \vee C(v))$$

は，G が頂点被覆を持つことを表わす．

MAXSNP$_0$ 問題を表わす論理式は，このような形の論理式である．MAX-CUT, MAX2SAT, MAX3SAT, 4-INDEPENDENT-SET それぞれについて，このような論理式を示せ．

[††] L.Kou, G.Markowsky and L.Berman, A fast algorithm for Steiner trees, *Acta Inform.* 15, pp.141-145, 1981.　その後，近似閾値はたかだか $\frac{5}{11}$ であることが次の論文で示された．

A.Z.Zelikowski, An 11/6-approximation algorithm for the network Steiner problem, *Algorithmica* 9, pp.463-470, 1993.

参 考 書 案 内

原則として1980年以前のものは省略した．最近出版されたものをすべて網羅しているわけではない．° 印を付けたものは，どれか1,2冊に絞るとしたとき推薦するものである．

本書全般は次の書物に多くを負っている．すぐれた研究者でもあるこれらの著者に敬意と謝意を表したい．

1. °J.E.Hopcroft and J.D.Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 1979, 418pp. (野崎昭弘・高橋正子・町田元・山崎秀記 (訳),「オートマトン 言語理論 計算論 I」, サイエンス社, 1984, 303pp.「同 II」, サイエンス社, 1986, 281pp.) [やや古いが，本書の5章までの内容をカバーする．この本の前半は，本書ではわずかしか触れていない形式言語・オートマトン理論に充てられている]
2. 笠井琢美,「計算量の理論」, 近代科学社, 1987, 236pp. [本書の5章までの内容をカバーする]
3. J.L.Balcazar, J.Diraz and J.Gabarro, *Structural Complexity I, II*, Springer-Verlag, 1988, 208pp. と 283pp. (I のみ邦訳あり: 足立暁生 (訳),「構造的計算量理論入門」, シュプリガーフェアラーク・東京, 1989.) [I, II ともに，やや専門家向けの記述である．II は本書がカバーしていない，相対化・Kolmogorov complexity・多項式時間同型写像・low hierarchy と high hierarchy 等も詳細に扱っている]
4. 笠井琢美・戸田誠之助,「計算の理論」, 共立出版, 1993, 225pp. [本書とほぼ同じ内容をカバー．計算モデルとして，TM ではなくプログラムを用いている]
5. °C.H.Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994, 523pp. [本書の内容と最もよくマッチしている．文献説明も詳しく，入門者から専門家にまで薦められる]
6. D.P.Bovet and P.Crescenzi, *Introduction to the Theory of Complexity*, Prentice-Hall, 1993, 282pp. [本書で扱っている内容全般を要領よく記述した入門書]

その他の参考書を項目別 (項目の中では出版年度順) に以下に列挙する．

計算量の理論全般

7. M.R.Garey and D.S.Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman and Co., 1979, 340pp. [**NP** 完全問題のバイブル]
8. 伊理正夫・野崎昭弘・野下浩平,「計算の効率化とその限界」, 日本評論社, 1980, 192pp. [今でも読む価値あり．本書で触れていない数値計算なども扱っている]
9. 小林孝次郎・高橋正子,「オートマトンの理論」, 共立出版, 1983, 258pp. [前半は有限オートマトン理論，後半は計算量理論]
10. A.Salomaa, *Computation and Automata*, Cambridge Univ. Press, 1985. (野崎昭弘・町田元・山崎秀記・横森貴 (訳),「計算論とオートマトン理論」, サイエンス社, 1988, 332pp.) [言語理論と計算量理論が中心，暗号系にやや詳しい]
11. 有川節夫・宮野悟,「オートマトンと計算可能性」, 培風館, 1986, 245pp. [言語理論＋本書の5章までをカバー]

12. K.Wagner and G.Wechsung, *Computational Complexity*, D.Reidel Publishing Co., 1986, 551pp. [専門家向け: 当時までの計算量理論の全結果のサーベイ]

13. 西野哲朗・足立暁生，「計算量理論概説」，朝倉書店，1988, 182pp. [本書の５章までの内容を扱っている]

14. 小林孝次郎，「計算の複雑さ」，昭晃堂，1989, 155pp. [**NP** の構造に詳しい]

15. J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, Elsevier, 1990. [専門家向けサーベイ．文献充実]

16. 足立暁生，「アルゴリズムと計算理論」，森北出版，1990, 200pp. [やさしい入門書]

17. 渡辺治，「計算可能性・計算の複雑さ入門」，近代科学社，1992, 226pp. [本書の５章までの内容を丁寧に解説している]

18. 岩田茂樹，「NP 完全問題入門」，共立出版，1995, 96pp. [**NP** 完全問題だけを扱ったユニークな入門書]

19. M.Sipser, *Introduction to the Theory of Computation*, PWS Publishing Co., 1997, 396pp. [前半が形式言語理論，後半が計算量理論に当てられている．形式的な証明の前に証明のアイデアの説明があり，入門書として薦められる]

計算論・計算可能性 (計算量理論の基礎１：本書第２章)

20. W.S.Brainerd and L.H.Landweber, *Theory of Computation*, John Wiley & Sons, 1974, 336pp. [loop プログラム・マルコフアルゴリズム・λ 計算など]

21. 細井勉，「計算の基礎理論」，教育出版，1975, 179pp.

22. 小林孝次郎，「計算可能性入門」，近代科学社，1980.

23. H.Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, The MIT Press, 1987, 482pp. [旧著 (1967) の改訂版]

24. 高橋正子，「計算論 — 計算可能性とラムダ計算」，近代科学社，1991, 191pp.

25. 渡辺治・米崎直樹，「計算論入門」，日本評論社，1997, 211pp.

オートマトン・言語理論 (計算量理論の基礎２：本書全般)

26. 嵩忠雄・都倉信樹・谷口健一，「形式言語理論」，電子情報通信学会，1988, 290pp.

27. 富田悦次・横森貴，「オートマトン・言語理論」，森北出版，1992, 205pp.

28. R.N.Moll, M.A.Arbib, A.J.Kfoury, *An Introduction to Formal Language Theory*, Springer-Verlag, 1988. (西野哲朗 (訳)，「形式言語理論入門」，東京電機大学出版局，1995, 298pp.)

並列アルゴリズム：本書第６章

29. M.J.Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, 1987, 288pp.

30. A.Gibbons and W.Rytter, *Efficient Parallel Algorithms*, Cambridge Univ. Press, 1988, 259pp.

31. S.G.Akl, *The Design and Analysis of Parallel algorithms*, Prentice-Hall, 1989.

32. °J.Jájá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992, 566pp.

33. °宮野悟,「並列アルゴリズム」, 近代科学社, 1993, 316pp. [並列アルゴリズムに関する本格的邦書としては唯一のもの]

34. R.Greenlaw, H.J.Hoover and W.L.Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*, Oxford Univ. Press, 1995, 311pp. [**P** 完全問題のリスト付き]

35. J.Hromkovič, *Communication Complexity and Parallel Computing*, Springer-Verlag, 1997, 336pp. [並列計算における通信計算量を研究する新分野の本格書]

離散数学: 本書第 1 章

36. M.A.Arbib, A.J.Kfoury and R.N.Moll, *A Basis for Theoretical Computer Science*, Springer-Verlag, 1981, 219pp. (甘利俊一他(訳),「計算機科学入門」, サイエンス社, 1984, 233pp.) [やさしい入門書]

37. 町田元・横森貴,「計算機数学」, 森北出版, 1990, 174pp.

38. °守屋悦朗,「コンピュータサイエンスのための離散数学」, サイエンス社, 1992, 227pp.

39. A.V.Aho and J.D.Ullman, *Foundations of Computer Science*, Computer Science Press, 1992, 765pp. [これもやさしい入門書]

その他

40. °石畑清,「アルゴリズムとデータ構造」, 岩波書店, 1989, 486pp.

41. °T.H.Cormen, C.E.Leiserson and R.L.Rivest, *Introduction to Algorithms*, The MIT Press, 1990, 1028pp. [やさしく書かれていながら, アルゴリズムに関するすべてについて述べたバイブル]

42. M.Behzad, G.Chartrand and L.Lesniak-Foster, *Graphs and Digraphs*, Prindle, Weber & Schmidt, 1979. (秋山仁・西関隆夫(共訳),「グラフとダイグラフの理論」, 共立出版, 1981, 434pp.)

43. C.H.Papadimitoriou and K.Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982. 496pp. [最適化問題のアルゴリズムと計算量]

44. E.L.Lawler, J.K.Lenstra 他, *The Traveling Salesman Problem*, John Wiley & Sons, 1985, 473pp. [巡回セールスマン問題だけをフィーチャ]

45. 池野信一・小山謙二,「現代暗号理論」, 電子情報通信学会, 1986, 289pp.

46. A.Salomaa, *Public-Key Cryptography*, Springer, 1990, 245pp. [公開鍵暗号系]

47. N.Alon, P.Erdös and J.H.Spencer, *The Probabilistic Method*, John Wiley & Sons, 1992, 254pp. [アルゴリズムの確率的手法に関するパラダイム集]

48. M.Li and P.Vitanyi, *Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, 1993, 546pp. [本書では述べなかった Kolmogorov complexity についての本格的入門書]

49. E.Mendelson, Introduction to Mathematical Logic, 4th ed., Chapman & Hall, 1997, 440pp. [1964 年初版の定本]

50. 福山克,「数理論理学」, 培風館, 1980, 228pp.

索　引

● さ　行

● た　行

● な　行

● は　行

● ま　行

• **X–Z**

• **ギリシャ文字**

• **記号**

Acknowledgement
The author's gratitude goes to his wife, Seiko, who has been giving him ease in their everyday life, which made it possible for this book to come into the world.

著者略歴

守屋悦朗（もりや えつろう）

1970年 早稲田大学卒業
1971年 早稲田大学大学院中退
1971年 電気通信大学助手
1976年 理学博士
1979年 東京女子大学専任講師
1989年 東京女子大学教授
1995年 早稲田大学教授
現 在 同上

主要著書

コンピュータサイエンスのための離散数学
(サイエンス社)

パソコンで数学（上・下）
(A.エンゲル著 共訳 共立出版)

1997年11月10日 初版発行

情報数理シリーズ B-2
チューリングマシンと計算量の理論

著 者 守屋悦朗
発行者 山本 格

発行所 株式会社 培風館
東京都千代田区九段南4-3-12・郵便番号102
電話(03)3262-5256(代表)・振替00140-7-44725

D.T.P. アベリー・平文社印刷・三水舎製本

PRINTED IN JAPAN

ISBN4-563-01492-3 C3355